

FIG. 2

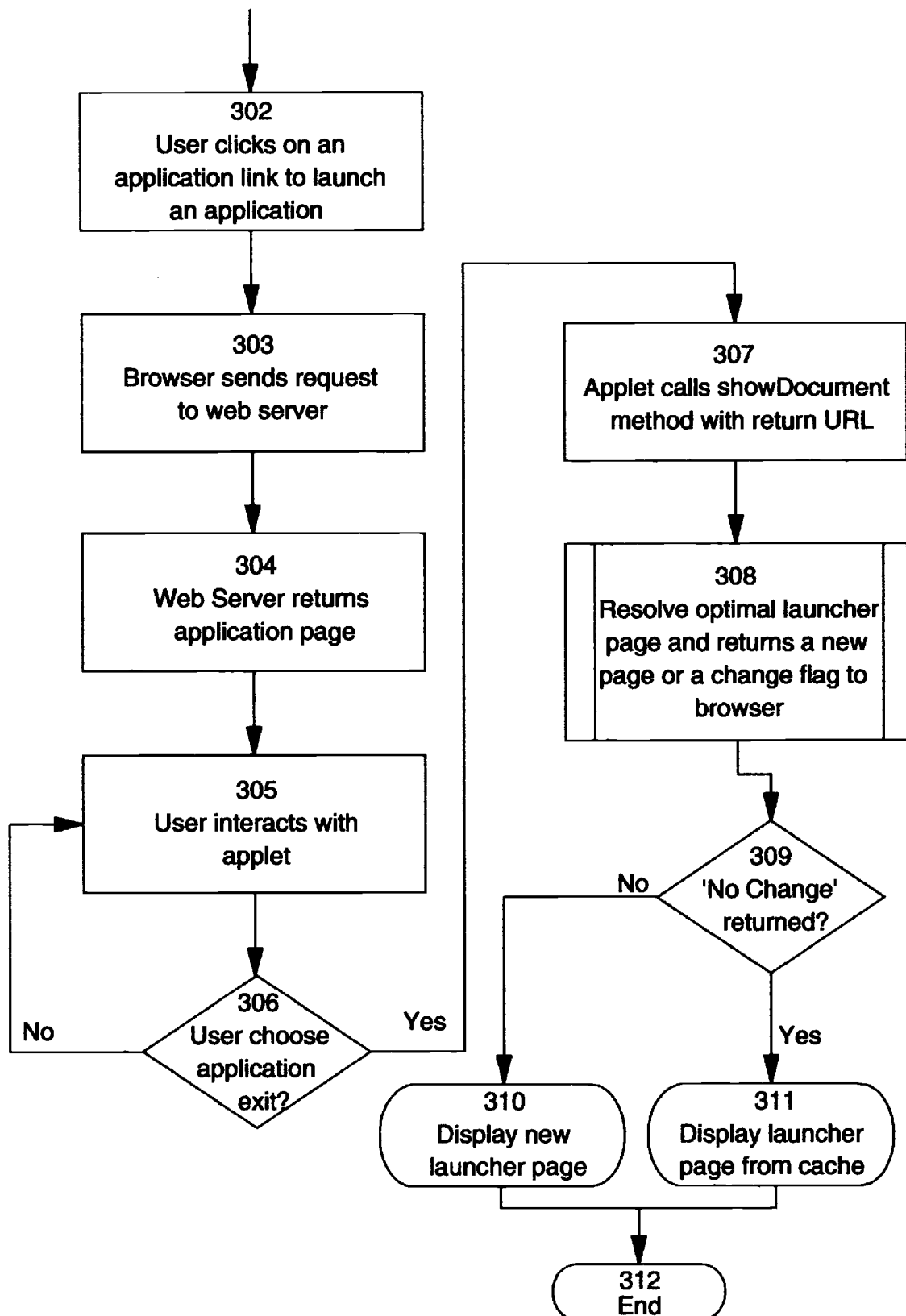


FIG. 3

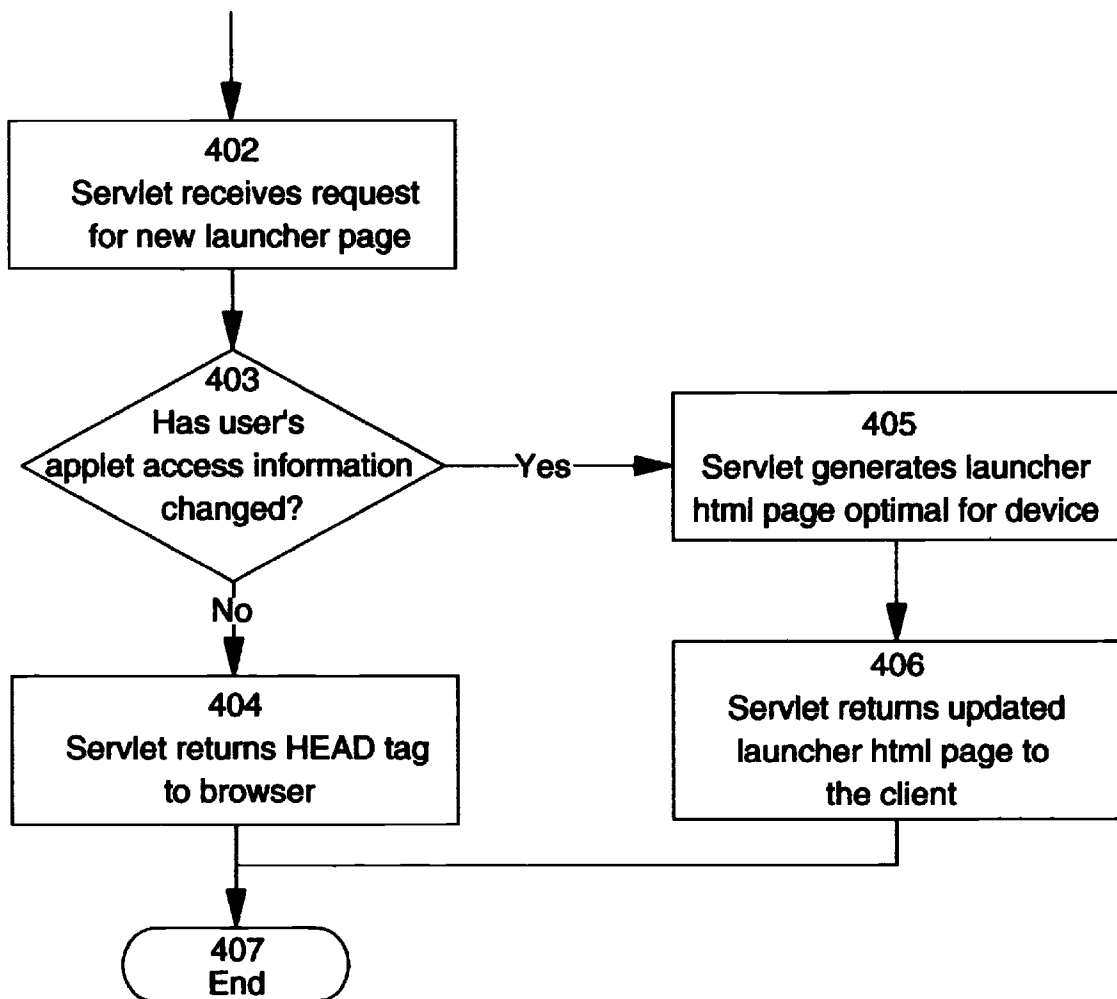


FIG. 4

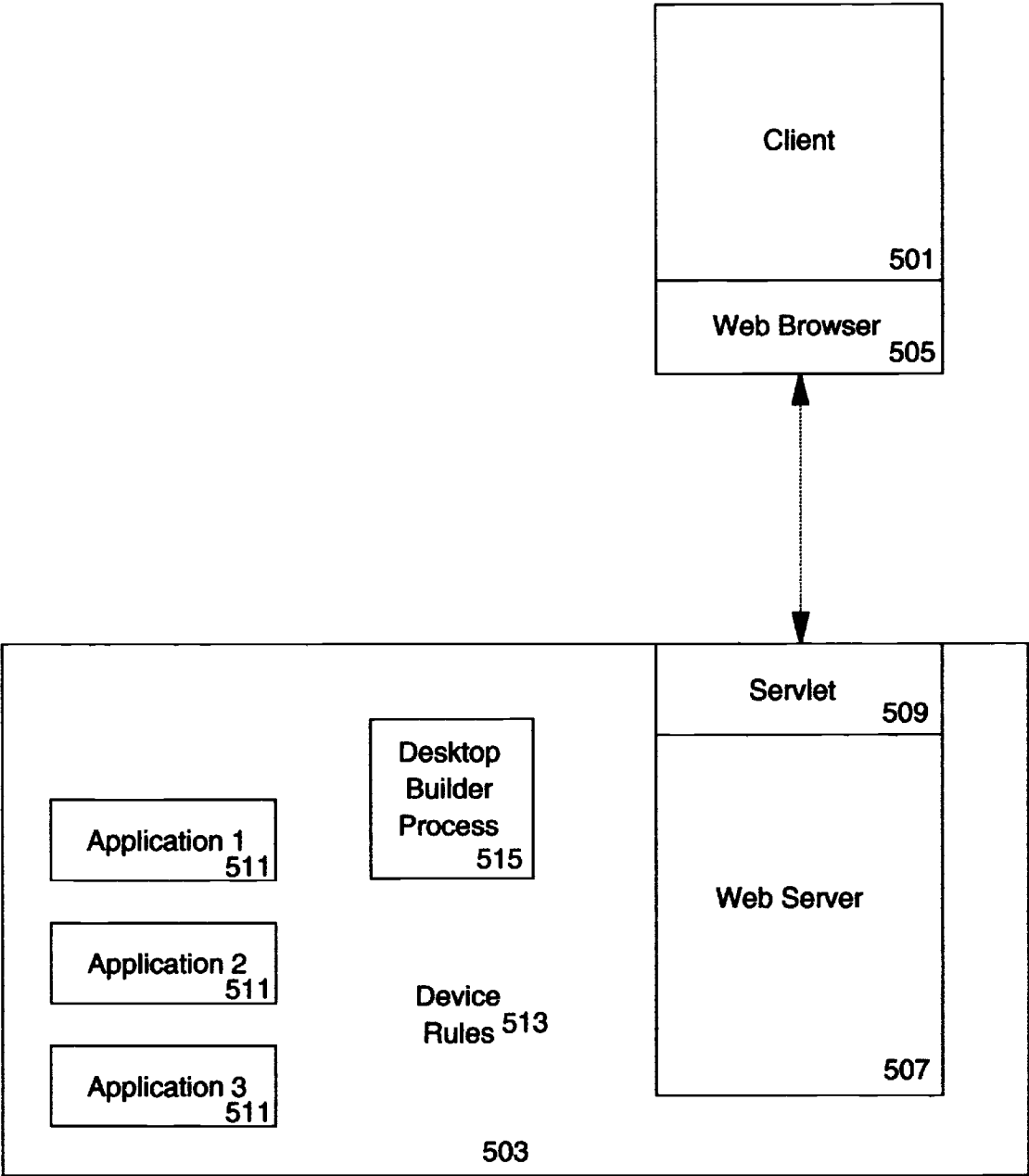


FIG. 5

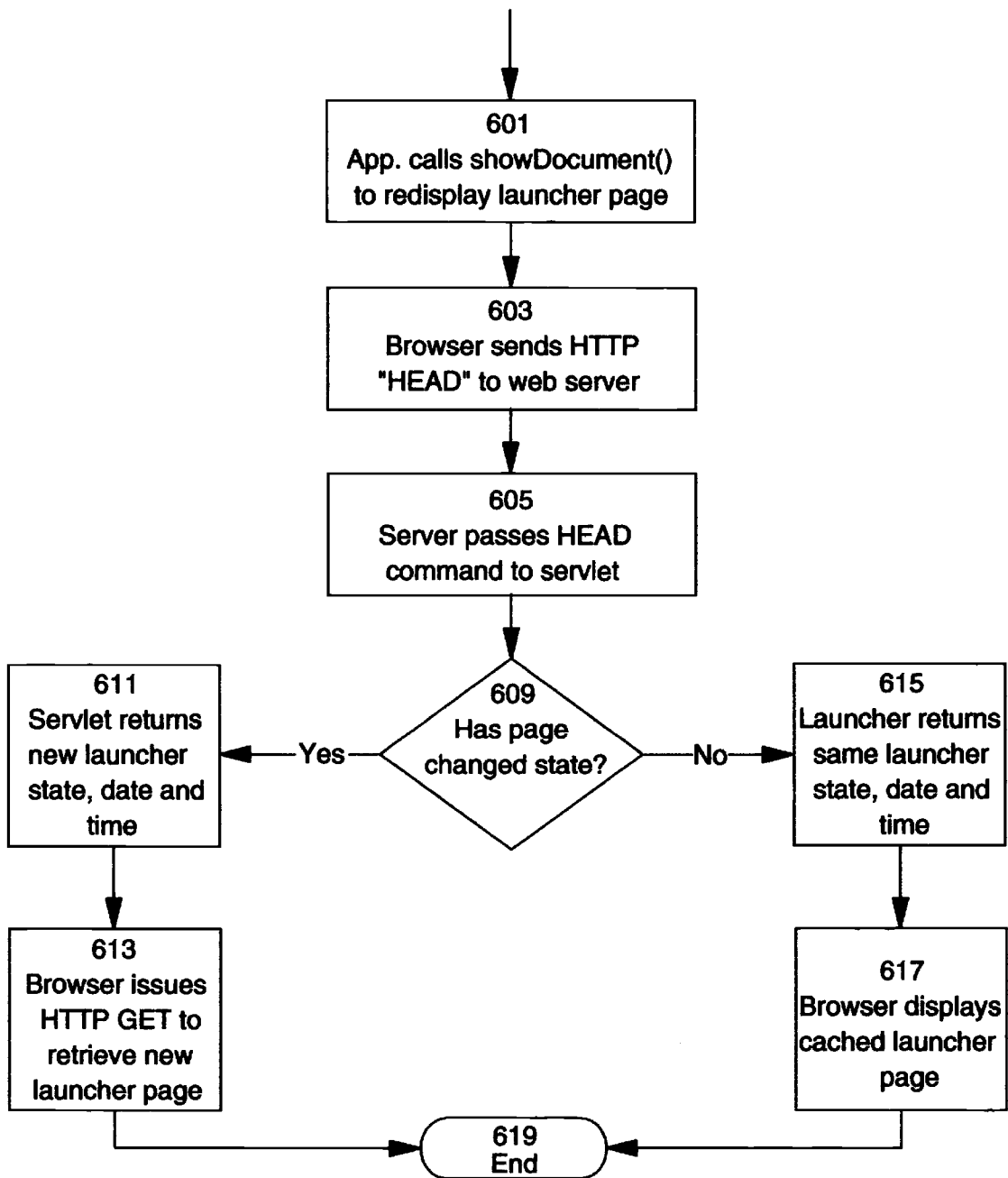


FIG. 6

US 6,212,564 B1

1

## DISTRIBUTED APPLICATION LAUNCHER FOR OPTIMIZING DESKTOPS BASED ON CLIENT CHARACTERISTICS INFORMATION

### BACKGROUND OF THE INVENTION

Computers have become pervasive in everyday life for many people. Computers are commonplace in the office and at home. Some people have hand-held computer devices to carry names and addresses with them, which connect to their notebook computer which they take to meetings and use when traveling, which also connect to their desktop unit for office or home use. The larger and more powerful computers usually are associated with physically larger units which are difficult for individuals to carry with them. This has resulted in several sizes of computers each having their own special purpose.

Users of computers become comfortable with applications and desire those applications to be portable across all of the machines which they use. A user of an address book on a small hand held device would also like to access that same address book on their office computer. They would rather not be required to learn a completely different address book. This is not so much a problem when the application is designed for the smallest computer, but it does become a problem when the application is designed for one of the larger, more robust units and the user wishes to execute the application on the smaller unit.

The ability to execute applications not specifically designed for a plurality of systems has exploded with the rapid acceptance of Java®<sup>1</sup> in the computing industry. Using Java, a computer user can download applications written in Java to their machine for execution. The user can request the execution of the application on their desktop system having extensive functionality as well as requesting the execution of the same application on a network computer having minimal functionality. This causes problems for robust applications. Java is a registered trademark of Sun Microsystems, Inc.

Several application designers have implemented license management applications wherein an application queries the server to ensure that the user is allowed to access the application which they desire to download for execution. In the design of these applications, the user's application data is typically stored on the server so that the user can access the application and data from any computer which they chose to logon to. The design of most of these products is vertical in nature, the system is designed for a specific subset of client devices rather than for all possible clients. This is partially due to the complexity and differences in capabilities across a broad range of devices. Even though most of the popular target devices have Java enabled HTML (hypertext mark-up language) browsers, the difference in function, bandwidth to the device and capability with these devices can vary widely therefore most solutions are limited to a subset of devices.

A need exists for a means of mitigating the functional differences between the different levels of computers with little or no impact to the end user. This would allow applications to be utilized on a wider range of devices. This is easier to understand through example. Take a 3270 emulator, for example. There could be three versions of an emulator named PC13270 available, a full function and fully enhanced graphical user interface version, a medium function and medium user interface version and a small function version. Each one is the same application to the user and has the same name to the user. The Applet Launcher link to the PC/3270 references the specific one based on the capabilities

2

of the device which is acquiring access. A small wireless unit might only be capable of executing the small version. From the user's perspective, it is the same application, perhaps with less fringe function, but the user can count on the core function, key mapping and other application attributes. This enables a series of applications that look the same in function and are named the same to execute across multiple platforms based on the specific capabilities of the platforms.

### SUMMARY OF THE INVENTION

The focal point of managed applications is the application launcher. This is the first interface that the user encounters when they start a system. From the launcher, the user selects applications and invokes them. The present invention provides a method, system and apparatus for providing a dynamic desktop construction component in a client/server based application suite. The construction component is added to the server that dynamically constructs the user's desktop based on the device from which the logon is received. This dynamically constructed desktop is optimized to provide a desktop matching the capabilities of the browser in the device, the capabilities of the Java runtime environment (JRE) in the device, the capabilities of the communication links in the device, the screen resolution and color depth of the device or any other system capabilities that the designer desires to optimize the application for. In addition, if the desktop is generated in HTML, it will allow the browser to query (via HTTP GET url HEAD) the page representing the desktop and the browser will automatically download the latest desktop when a new desktop becomes available.

### BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a flow chart representative of the client logic of the present invention.

FIG. 2 is a flow chart representative of the server logic for the present invention.

FIG. 3 is a flow chart of the communications interaction between the client and the server of the present invention.

FIG. 4 is a flow chart representing the page update process of the present invention.

FIG. 5 is a block diagram of the components of the preferred embodiment of the present invention.

FIG. 6 is a further revision of the flow chart representing the preferred embodiment of the present invention.

### DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention will now be discussed in greater detail with reference to the attached figures and a preferred embodiment. The embodiment of the present invention may be implemented in software or as a combination of hardware and software such as smart cards. In the preferred embodiment the launcher is implemented as an applet launchers although other embodiments of the launcher are envisioned. The preferred embodiment is not meant to limit the scope of the invention in any way but only to provide a workable example.

Referring to FIG. 5, the present invention implements a client process that is run on a distributed computer 501 and a server process running on a host or server computer 503. When a user requests a url (uniform resource locator) by way of a web browser 505, the browser sends a request to a web server 507. The web server of the present invention will pass the request to a servlet 509 for processing. The

US 6,212,564 B1

3

servlet **509** returns to the client **501**, via the web server **507**, the application launcher for the client based on device information passed to the servlet in the request. The servlet uses a set of rules **513** to build the optimal application launcher based on the capabilities of the client and, if desired, the network bandwidth. These capabilities of the client include, but are not limited to, the capabilities of the browser running on the client, the resolution and capabilities of the monitor and hardware executing the browser and any other information relevant to the execution of applications or applets on the client.

The present invention allows a user to invoke an instance of a product or a suite of products from one device and receive a desktop customized for that device with the associated set of applications that will run efficiently on that device and subsequently invoke the product or suite on a different device and receive a different desktop of the same set of applications that is tailored to run efficiently on the subsequent device. This gives a provider of applications or application suites the flexibility to utilize the robust features of larger, more sophisticated machines yet still support the limited functionality of smaller or less sophisticated devices. For example, a suite provider can choose to provide highly functional but larger sized Java applications to the larger, higher function machines and still deliver smaller, limited function versions of the applications written in Java or other languages such as Javascript and DHTML to a machine with less functionality and less space. The customization of these applications includes not only functionality but includes image and color definitions for the desktop, screen resolution and sound capabilities for example. The customization of the image color and depth is especially important when dealing with the differences between enhanced color screens and monochrome or gray-scale screens.

In the preferred embodiment of the present invention, a client process is utilized to enable the user to view an application launching desktop. The client process is typically written as a Java applet or as HTML (hyper-text markup language) pages. When an application is terminating it can redisplay the desktop using the showDocument() function which is available in HTML via Javascript or in Java applets from the showDocument() method in the applet class. A server process is responsible for providing the proper desktop to the client using HTTP. This can be done using standard HTTP request protocols. In the preferred embodiment, when the server process rebuilds the user's page the authoring date within the page is updated to indicate the current date and time. This causes the browser to load the page from the server rather than rereading the page from the browser's cache if the page has been updated since the time indicated on the copy in the browser's cache.

Referring to the figures, FIG. 1 depicts the flow of the logic in the browser process of the preferred embodiment of the present invention. First the user invokes the web browser **102** which resides on the client. The user then issues the commands to enter the Java launcher logon page which is a URL **103**. The browser then sends an HTTP request for the logon page to the server **104**. The browser then waits to receive the launcher logon page from the server **105**. When the browser receives the launcher logon page, it displays the page **106** on the user's display device. The user then enters a user identifier, password (if required) and a device type **107** (if not automated). Using HTTP, the browser then sends a request for an applet launcher page along with the user identifier, password and device type to the server **108**. As will be obvious to one skilled in the art, several devices have the ability to provide their device type rather than querying

4

the user. Java included in the logon page of the preferred embodiment will attempt to identify the device or prompt the user for the device if it cannot be determined programmatically. This reduces the number of errors introduced by misidentification of the system type entered manually by a user. Once the browser has sent the request for an applet launcher page, the web server receives the request and transfers the request to a dynamic desktop servlet residing on the server. The servlet retrieves/constructs the optimal launcher page and returns the page to the user's browser. The user's browser receives the optimal launcher page **110** and displays the optimal applet launcher page **111** on the display of the requesting device.

FIG. 2 depicts the logic flow on the server side of the process. First a servlet running on the server receives the request for the applet launcher page based on the user identifier, password and device type **202**. The servlet validates the user identifier and password **203** and if they are valid the servlet generates or selects the applet launcher HTML page which is optimal for the requested device **204**. The servlet then returns the applet launcher page to the client **205**.

If, at **203**, it was determined that either the user identifier or password were invalid, the servlet would return an invalid user identifier/password page to the client **206**. The browser on the client device would receive the page and display it. If the user were to re-enter the user identifier and password, the browser could then send the updated page to the server and the servlet would receive the request for the applet launcher page again **202**.

FIG. 3 depicts the logical information flow between the browser and the server in the preferred embodiment of the present invention. First the user selects an application link to launch an application **302**. The browser then, using HTTP, sends the request to the web server for the selected application **303**. The web server returns an application page for the selected application to the web browser **304**, the application page containing a selected Java applet. The user is then able to interact with the applet **305**. The browser then waits until the user chooses to exit the selected application **306**, when the user chooses to exit, the applet calls the showDocument method with the return URL **307**. The servlet then resolves the optimal application launcher page and returns a new page or an HTTP HEAD tag indicating "no change" to the user's browser **308**. The browser checks to see if a "no change" indication has been returned **309**. If it has been returned then the browser displays the applet launcher page from the browser's cache **311**, otherwise the browser displays the new applet launcher page received from the servlet **310**.

FIG. 4 further describes the process of resolving the optimal applet launcher page. At **401** the servlet, running on the server, receives a request for an applet launcher page. A test is made to determine whether the user's device has changed **402**. If the information has not changed, a test is made to determine whether the user's applet access information has changed **403**. If the information has not changed then the servlet returns an HTTP indicator depicting "no change" to the browser **404**. If, at **403**, the user's applet access information has changed or at **402** the device has changed, the servlet generates an applet launcher HTML page optimal for the indicated device **405** based on the screen resolution, depth of color, device type and application set. The servlet then returns the updated applet launcher HTML page to the client **406**.

FIG. 6 is a further refinement of the decision process to display a new launcher page for the preferred embodiment



US 6,212,564 B1

5

of the present invention. At 601 the executing application calls the showDocument( ) command to redisplay the launcher page. The browser then sends an HTTP "HEAD" request 603 to the web server indicating the metainformation for the indicated resource. The server passes the HEAD 5 command to the servlet for processing 605. The servlet determines if the launcher page has changed state 609 from the metainformation received. If the launcher page has changed state, the servlet returns the new launcher state, date and time indications 611, then the browser issues an HTTP 10 GET to retrieve the new launcher page 613. If the launcher page has not changed state, then the launcher returns the same launcher state, date and time indications 615. This indicates to the browser to redisplay the cached launcher page 617.

What is claimed is:

1. A system for customizing applications delivered to a client computer in a client/server network, said client computer having characteristic information, said system comprising:

a server computer;  
an application suite residing on said server computer;  
a web server residing on said server computer;  
a web browser residing on said client computer; and,  
a servlet associated with said web server, said servlet receiving information indicating said client characteristics of said client computer, constructing, by said servlet, a desktop optimized for said client characteristics of said client computer and returning said optimized desktop to said web browser residing on said client computer.

2. A method for customizing applications delivered to a client computer in a client/server network, said client computer having characteristic information, said network comprising a server computer, an application suite residing on said server computer, a web server residing on said server computer and a web browser residing on said client computer, said method comprising:

means associated with said web server for receiving information indicating the client characteristics of said client computer;

means, responsive to receiving said information, for constructing a desktop optimal for said characteristics of said client computer; and,

means for returning said optimal desktop to said web browser residing on said client computer.

3. A program residing on a computer readable medium for customizing applications delivered to a client computer in a client/server network, said client computer having client characteristic information, said network comprising a server computer, an application suite residing on said server computer, a web server residing on said server computer and a web browser residing on said client computer, said program comprising:

computer executable program code means associated with said web server for receiving said client characteristic information of said client computer;

computer executable program code means, responsive to receiving said client characteristic information, for constructing a desktop optimal for said characteristics of said client computer; and,

computer executable program code means for returning said optimal desktop to said web browser residing on said client computer.

6

4. An apparatus for customizing applications delivered to a client computer in a client/server network, said client computer having characteristic information, said apparatus comprising:

computer readable program code associated with a web server, said computer readable program code associated with said web server for receiving information indicating the characteristics of said client computer, constructing a desktop optimal for said characteristics of said client computer and transmitting said optimal desktop to said client computer.

5. A system as in claim 1 wherein said client/server network has network characteristic information wherein said servlet receives information indicating said network characteristics and utilizes both said client characteristics and said network characteristics in constructing said optimized desktop.

6. A method for customizing applications delivered to a client computer in a client/server network, said client computer having client characteristic information, said network comprising a server computer, an application suite residing on said server computer, a web server residing on said server computer and a web browser residing on said client computer, said network having network characteristic information said method comprising:

means associated with said web server for receiving information indicating said client characteristic information;

means associated with said web server for receiving information indicating said network characteristic information;

means, responsive to receiving said information indicating said client characteristics and said information indicating said network characteristics, for constructing a desktop optimal for said characteristics of said client computer; and,

means for returning said optimal desktop to said web browser residing on said client computer.

7. A program residing on a computer readable medium for customizing applications delivered to a client computer in a client/server network, said client computer having client characteristic information and said network having network characteristic information, said network comprising a server computer, an application suite residing on said server computer, a web server residing on said server computer and a web browser residing on said client computer, said program comprising:

computer executable program code means associated with said web server for receiving said client characteristic information of said client computer;

computer executable program code means associated with said web server for receiving said network characteristic information;

computer executable program code means, responsive to receiving said client characteristic information and said network characteristic information, for constructing a desktop optimal for said characteristics of said client computer and said network; and,

computer executable program code means for returning said optimal desktop to said web browser residing on said client computer.

\* \* \* \* \*



(12) **United States Patent**  
**Landsman et al.**

(10) **Patent No.:** **US 6,317,761 B1**  
(45) **Date of Patent:** **Nov. 13, 2001**

(54) **TECHNIQUE FOR IMPLEMENTING BROWSER-INITIATED USER-TRANSPARENT ADVERTISING AND FOR INTERSTITIALLY DISPLAYING AN ADVERTISEMENT, SO DISTRIBUTED, THROUGH A WEB BROWSER IN RESPONSE TO A USER CLICK-STREAM**

**FOREIGN PATENT DOCUMENTS**

0 818 742 1/1998 (EP) .  
0 822 535 A2 2/1998 (EP) .  
0 903 903 A2 3/1999 (EP) .

(List continued on next page.)

**OTHER PUBLICATIONS**

(75) Inventors: **Rick W. Landsman**, Waccabuc;  
**Wei-Yeh Lee**, New York, both of NY (US)

Kipp Cheng, "Rich Media Match for Unicast and Engage", Brandweek, Nov. 1, 1999, vol. 40, No. 41, p. 62 and following.

(73) Assignee: **Unicast Communications Corporation**, New York, NY (US)

(List continued on next page.)

(\*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

*Primary Examiner*—Joseph H. Field

(74) *Attorney, Agent, or Firm*—Michaelson & Wallace, Peter L. Michaelson

(21) Appl. No.: **09/352,398**

(57) **ABSTRACT**

(22) Filed: **Jul. 13, 1999**

**Related U.S. Application Data**

(60) Division of application No. 09/237,718, filed on Jan. 26, 1999, which is a continuation-in-part of application No. 09/080,165, filed on May 15, 1998, now abandoned.

(51) **Int. Cl.**<sup>7</sup> ..... **G06F 17/21**

(52) **U.S. Cl.** ..... **707/513; 705/27; 709/231**

(58) **Field of Search** ..... **707/513, 501; 709/203, 217–219, 230–235; 705/1, 10, 14, 26, 27**

(56) **References Cited**

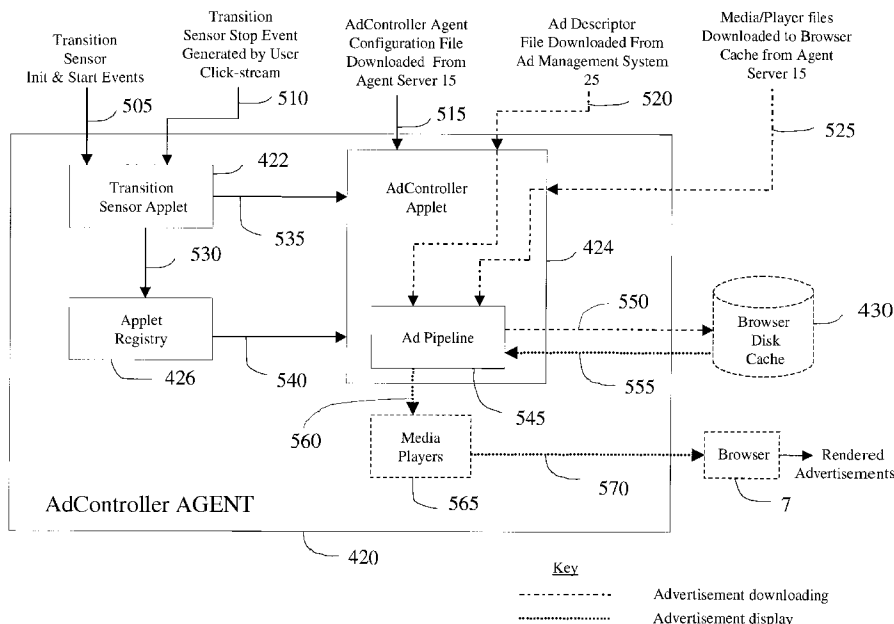
**U.S. PATENT DOCUMENTS**

4,575,579 3/1986 Simon et al. .... 178/4  
4,719,567 1/1988 Whittington et al. .... 364/200

(List continued on next page.)

A technique for implementing in a networked client-server environment, e.g., the Internet, network-distributed advertising in which advertisements are downloaded, from an advertising server to a browser executing at a client computer, in a manner transparent to a user situated at the browser, and subsequently displayed, by that browser on an interstitial basis, in response to a click-stream generated by the user to move from one web page to the next. Specifically, an HTML advertising tag is embedded into a referring web page. This tag contains two components. One component effectively downloads, from a distribution web server and to an extent necessary, and then persistently instantiates an agent at the client browser. This agent "politely" and transparently downloads advertising files, for a given advertisement into browser cache and subsequently plays those media files through the browser on an interstitial basis and in response to a user click-stream.

**72 Claims, 26 Drawing Sheets**



US 6,317,761 B1

Page 2

U.S. PATENT DOCUMENTS						
4,775,935	10/1988	Yourick .....	364/401	5,664,948	9/1997 Dimitriadis et al. .... 434/307 R	
4,782,449	11/1988	Brinker et al. ....	364/473	5,701,451	12/1997 Rogers et al. .... 395/600	
4,799,146	1/1989	Chauvel .....	364/200	5,706,502	1/1998 Foley et al. .... 395/610	
4,850,007	7/1989	Marino et al. ....	379/67	5,710,918	1/1998 Lagarde et al. .... 395/610	
5,027,400	6/1991	Baji et al. ....	380/20	5,717,860	2/1998 Graber et al. .... 395/200.12	
5,029,104	7/1991	Dodson et al. ....	364/514	5,721,827	2/1998 Logan et al. .... 395/200.47	
5,093,718	3/1992	Hoarty et al. ....	358/84	5,724,521	3/1998 Dedrick .....	395/226
5,099,420	3/1992	Barlow et al. ....	395/325	5,737,619	4/1998 Judson .	
5,105,184	4/1992	Pirani et al. ....	340/721	5,737,739	4/1998 Shirley et al. ....	707/512
5,159,669	10/1992	Trigg et al. ....	395/159	5,740,549	4/1998 Reilly et al. ....	705/14
5,165,012	11/1992	Crandall et al. ....	395/100	5,742,768	4/1998 Gennaro et al. ....	295/200.33
5,220,420	6/1993	Hoarty et al. ....	358/86	5,754,830	5/1998 Butts et al. ....	395/500
5,220,516	6/1993	Dodson et al. ....	364/514	5,761,601	6/1998 Nemirofsky et al. ....	455/3
5,220,564	6/1993	Tuch et al. ....	370/94.1	5,768,508	6/1998 Eikeland .....	395/200.32
5,253,341	10/1993	Rozmanith et al. ....	395/200	5,781,894	7/1998 Petrecca et al. .	
5,268,963	12/1993	Monroe et al. ....	380/23	5,787,254	7/1998 Maddalozzo, Jr. et al. ....	395/200.58
5,283,639	2/1994	Esch et al. ....	348/6	5,794,210	8/1998 Goldhaber et al. ....	705/14
5,283,731	2/1994	Lalonde et al. ....	364/401	5,796,952	8/1998 Davis et al. ....	395/200.54
5,285,442	2/1994	Iwamura et al. ....	370/17	5,805,815	9/1998 Hill .....	395/200.48
5,297,249	3/1994	Bernstein et al. ....	395/156	5,809,242	9/1998 Shaw et al. ....	395/200.47
5,305,195	4/1994	Murphy .....	364/401	5,809,481	9/1998 Baron et al. ....	705/14
5,307,456	4/1994	MacKay .....	395/154	5,838,458	11/1998 Tsai .....	358/402
5,313,455	5/1994	van der Wal et al. ....	370/13	5,854,897	12/1998 Radziewicz et al. .	
5,319,455	6/1994	Hoarty et al. ....	348/7	5,864,823	1/1999 Levitan .....	105/14
5,321,740	6/1994	Gregorek et al. ....	379/96	5,870,769	2/1999 Freund .....	707/501
5,325,423	6/1994	Lewis .....	379/90	5,897,622	4/1999 Blinn et al. ....	705/26
5,325,483	6/1994	Ise et al. ....	395/162	5,913,040	6/1999 Rakavy et al. ....	395/200.62
5,327,554	7/1994	Palazzi et al. ....	395/600	5,918,012	6/1999 Astiz et al. ....	395/200.47
5,333,237	7/1994	Stefanopoulous et al. ....	395/12	5,918,214	6/1999 Perkowski .....	705/27
5,347,632	9/1994	Filepp et al. .		5,923,853	7/1999 Danneels .....	395/200.68
5,355,472	10/1994	Lewis .....	395/600	5,931,907	8/1999 Davies et al. ....	709/218
5,355,501	10/1994	Gross et al. ....	395/750	5,933,811	8/1999 Angles et al. ....	705/14
5,361,091	11/1994	Hoarty et al. ....	348/7	5,937,390	8/1999 Hyodo .....	705/14
5,361,199	11/1994	Shoquist et al. ....	364/401	5,937,392	8/1999 Alberts .....	705/14
5,361,393	11/1994	Rossillo .....	395/650	5,937,411	8/1999 Becker .	
5,367,621	11/1994	Cohen et al. ....	395/154	5,946,646	8/1999 Schena et al. .	
5,392,447	2/1995	Schlack et al. ....	395/800	5,946,664	8/1999 Ebisawa .....	705/14
5,412,720	5/1995	Hoarty .....	380/15	5,946,697	8/1999 Shen .....	707/104
5,418,549	5/1995	Anderson et al. ....	345/145	5,948,061	9/1999 Merriman et al. ....	709/219
5,438,518	8/1995	Bianco et al. ....	364/460	5,960,409	9/1999 Wexler .....	705/14
5,442,771	8/1995	Filepp et al. ....	395/650	5,961,602	10/1999 Thompson et al. ....	709/229
5,483,466	1/1996	Kawahara et al. ....	364/514 C	5,963,909	10/1999 Warren et al. ....	705/1
5,491,785	2/1996	Robson et al. ....	395/162	5,978,841	11/1999 Berger .....	709/217
5,500,890	3/1996	Rogge et al. ....	379/91	5,978,842	11/1999 Noble et al. ....	709/218
5,515,098	5/1996	Carles .....	348/8	5,983,244	11/1999 Nation .....	707/501
5,515,270	5/1996	Weinblatt .....	364/405	5,983,268	11/1999 Freivald et al. ....	709/218
5,515,490	5/1996	Buchanan et al. ....	395/154	5,987,466	11/1999 Greer et al. ....	707/10
5,524,195	6/1996	Clanton, III et al. ....	395/155	5,991,799	11/1999 Yen et al. ....	709/218
5,524,197	6/1996	Uya et al. ....	395/157	5,996,007	11/1999 Klug et al. ....	709/218
5,530,472	6/1996	Bregman et al. ....	348/15	5,999,740	12/1999 Rowley .	
5,532,735	7/1996	Blahut et al. ....	348/13	5,999,912	12/1999 Wodarz et al. ....	705/14
5,541,986	7/1996	Hou .....	379/201	6,009,410	12/1999 LeMole et al. ....	705/14
5,548,745	8/1996	Egan et al. ....	395/500	6,011,537	1/2000 Slotznick .....	345/115
5,563,804	10/1996	Mortensen et al. ....	364/514 A	6,012,083	1/2000 Savitzky et al. .	
5,564,043	10/1996	Siefert .....	395/600	6,014,698	1/2000 Griffiths .....	709/224
5,572,643	11/1996	Judson .		6,035,332	3/2000 Ingrassia, Jr. et al. .	
5,579,381	11/1996	Courville et al. ....	379/201	6,046,252	12/1999 Wolfe .	
5,583,560	12/1996	Florin et al. ....	348/7	6,047,318	4/2000 Becker et al. .	
5,590,046	12/1996	Anderson et al. ....	364/474.13	6,061,659	5/2000 Murray .....	705/14
5,594,509	1/1997	Florin et al. ....	348/731	6,091,411	7/2000 Straub et al. .	
5,594,779	1/1997	Goodman .....	379/59	6,112,246	8/2000 Horbal et al. .	
5,596,718	1/1997	Boebert et al. ....	395/187.01	6,119,165	9/2000 Li et al. .	
5,602,905	2/1997	Mettke .....	379/96	6,125,388	9/2000 Reisman .	
5,606,359	2/1997	Youden et al. ....	348/7	6,157,946	12/2000 Itakura et al. ....	709/217
5,615,131	3/1997	Mortensen et al. ....	364/514 A	6,167,453	12/2000 Becker et al. .	
5,621,456	4/1997	Florin et al. ....	348/7	6,185,586 *	2/2001 Judson .....	707/513
5,629,978	5/1997	Blumhardt et al. ....	379/201	FOREIGN PATENT DOCUMENTS		
5,630,081	5/1997	Rybicki et al. ....	395/348	2784254A	4/2000 (FR) .	
5,635,979	6/1997	Kostreski et al. ....	348/13	9-114781	5/1997 (JP) .	
5,657,450	8/1997	Rao et al. ....	395/610	10-312344	11/1998 (JP) .	

US 6,317,761 B1

Page 3

11-003072 1/1999 (JP) .  
11-154159 6/1999 (JP) .  
97-78058 12/1997 (KR) .  
WO 96/30864 10/1996 (WO) .  
WO 97/07656 3/1997 (WO) .  
WO97/10502 4/1997 (WO) .  
WO 97/21183 6/1997 (WO) .  
WO 98/25198 6/1998 (WO) .  
WO 99/09486 2/1999 (WO) .

OTHER PUBLICATIONS

Cartellieri et al., “The Real Impact of Internet Advertising”, McKinsey Quarterly 1997, 1997, vol. 3, p. 3 of reprint.  
C. Taylor, “Going Beyond the Banner”, *Adweek Marketing Week* (now *Brandweek*), Interactive Quarterly Section, Jul. 8, 1996, pp. 22 et seq (downloaded as six pages, specifically pp. 36–41).

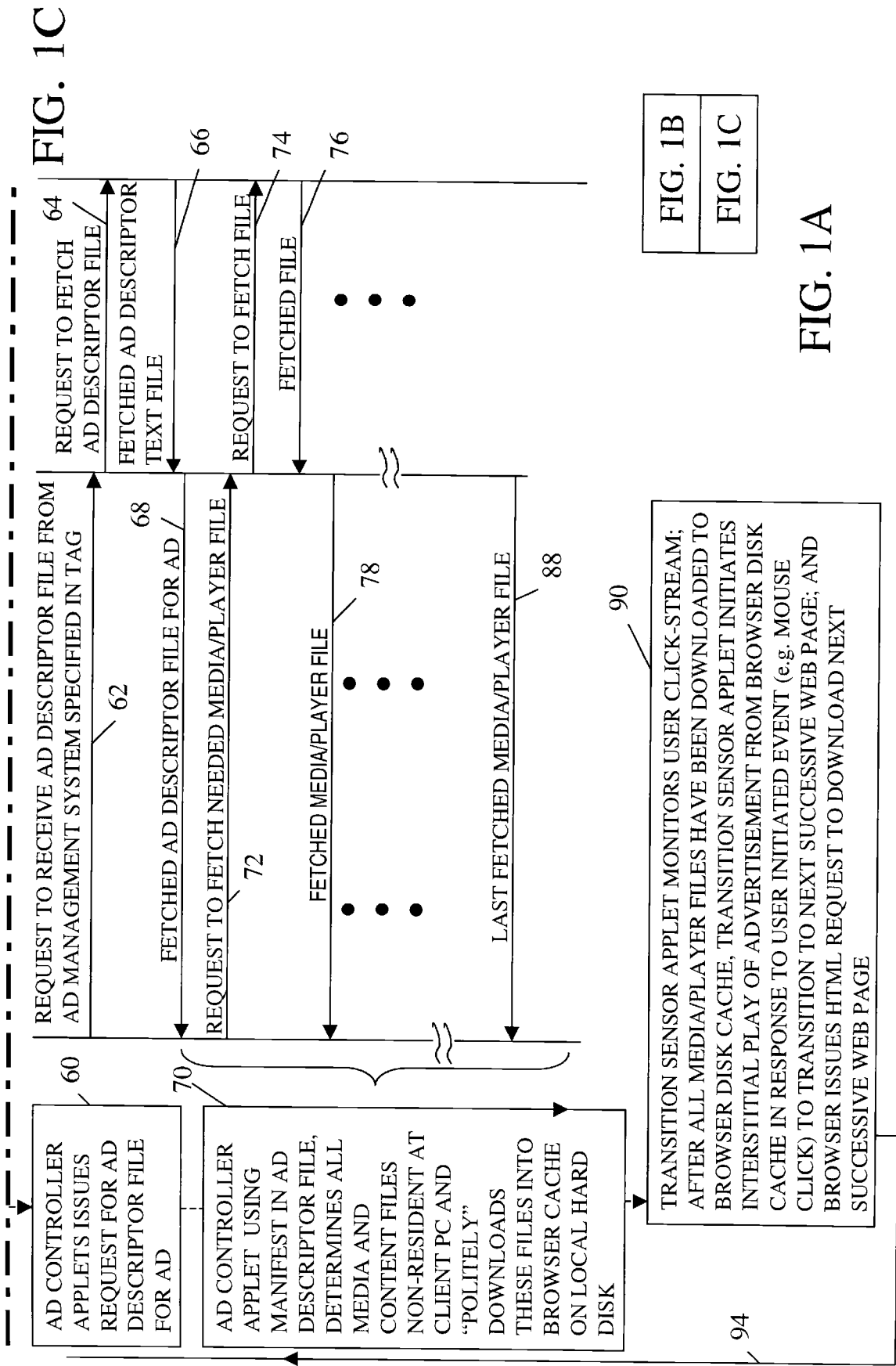
J. Hodges, “Marketers play web games as serious biz”, *Advertising Age*, Interactive Media and Marketing section, Mar. 14, 1996, one page.

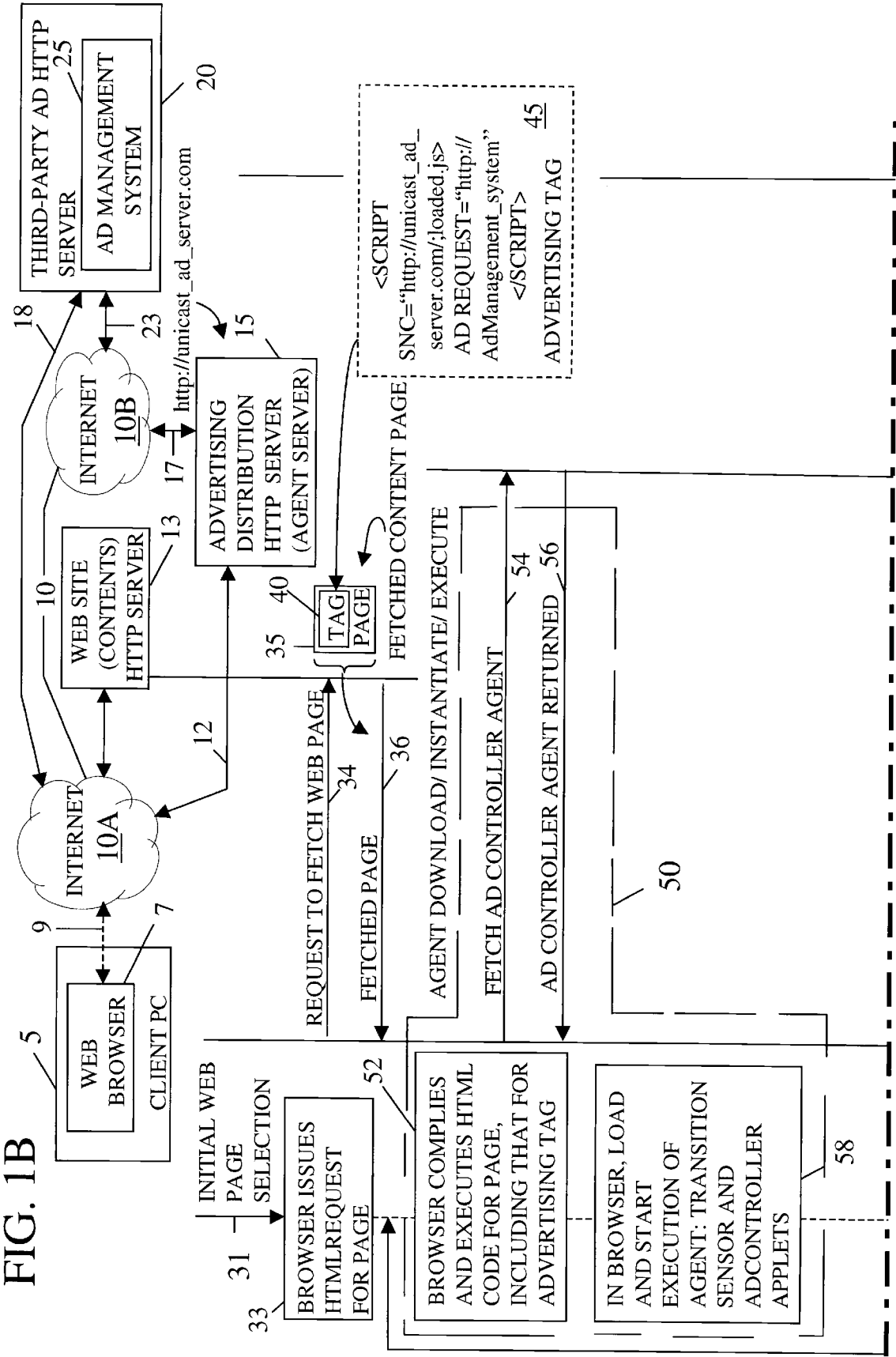
“Web Ads Start To Click”, *Business Week*, Oct. 6, 1997, pp. 128, 130–132, 134 and 138.

P. van der Linden, *Just Java 1.2—fourth Edition*, (© 1999, Sun Microsystems, Inc.), specifically Chapter 13, “All About Applets”, pp. 322–346.

Y. Kohda et al, “Ubiquitous Advertising on the WWW: Merging Advertisement on the Browser”, *Computer Networks and ISDN Systems* 28, Elsevier Science B.V., 1996, pp. 1493–1499.

\* cited by examiner





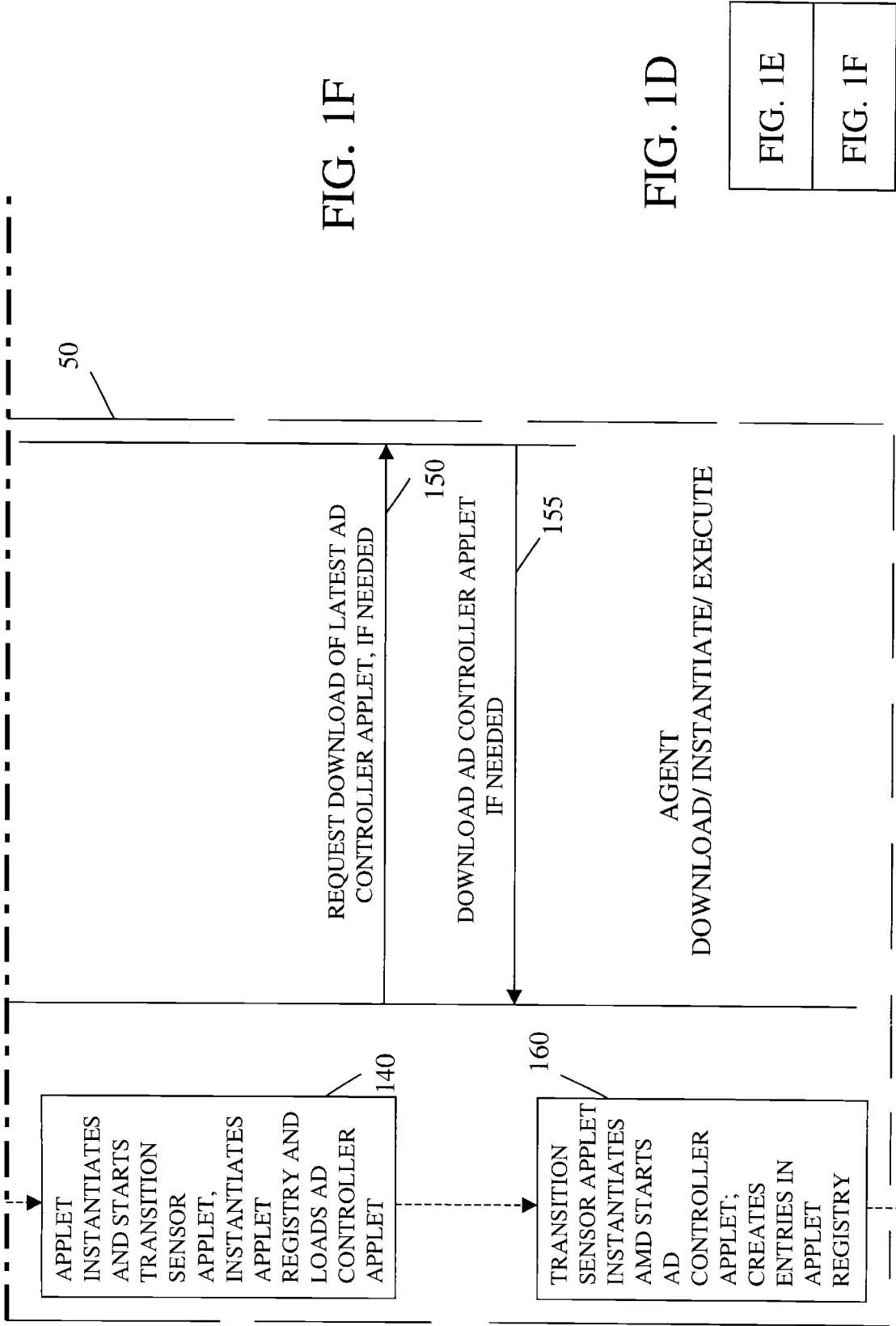


FIG. 1E

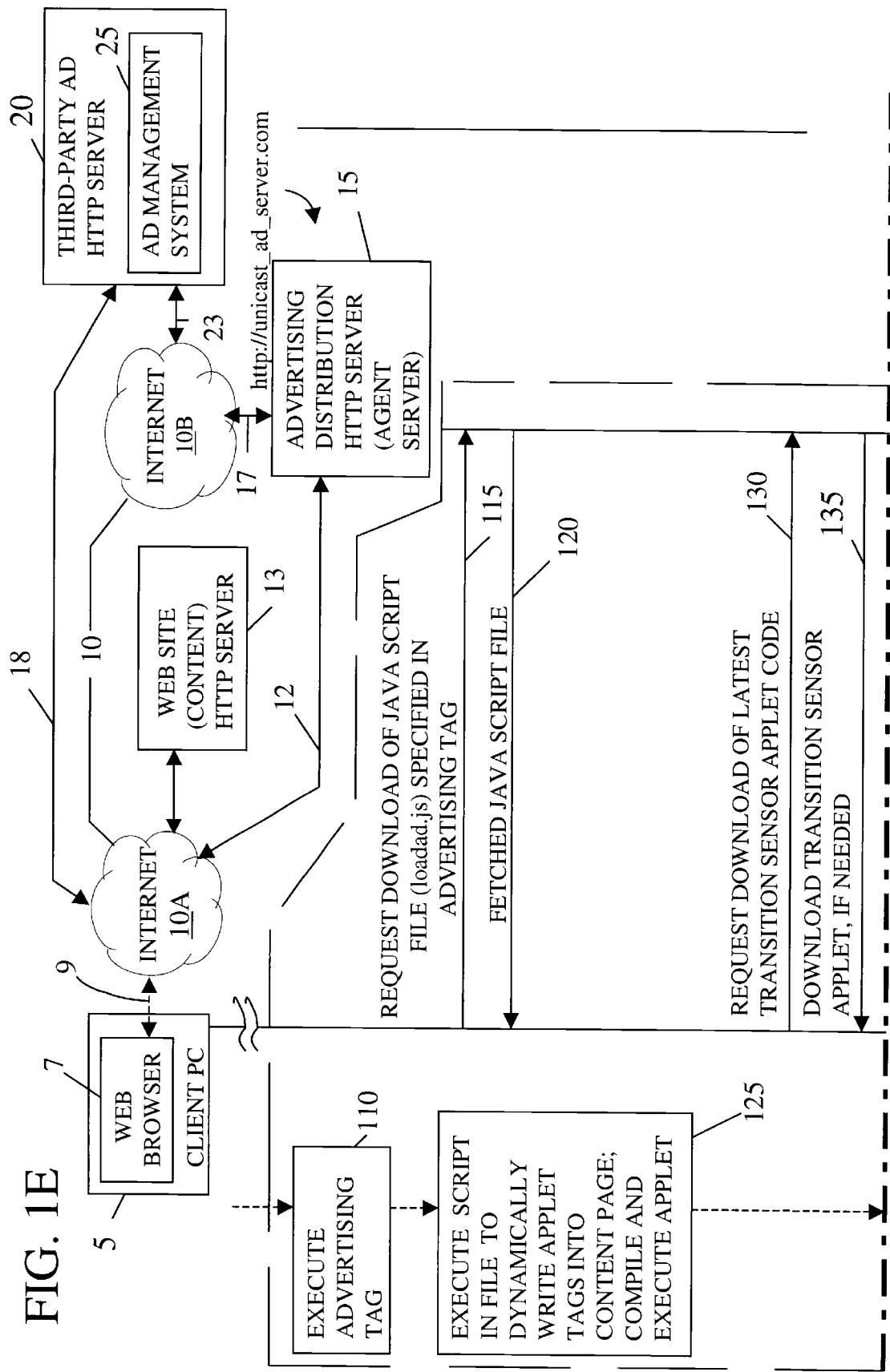




FIG.  
2A

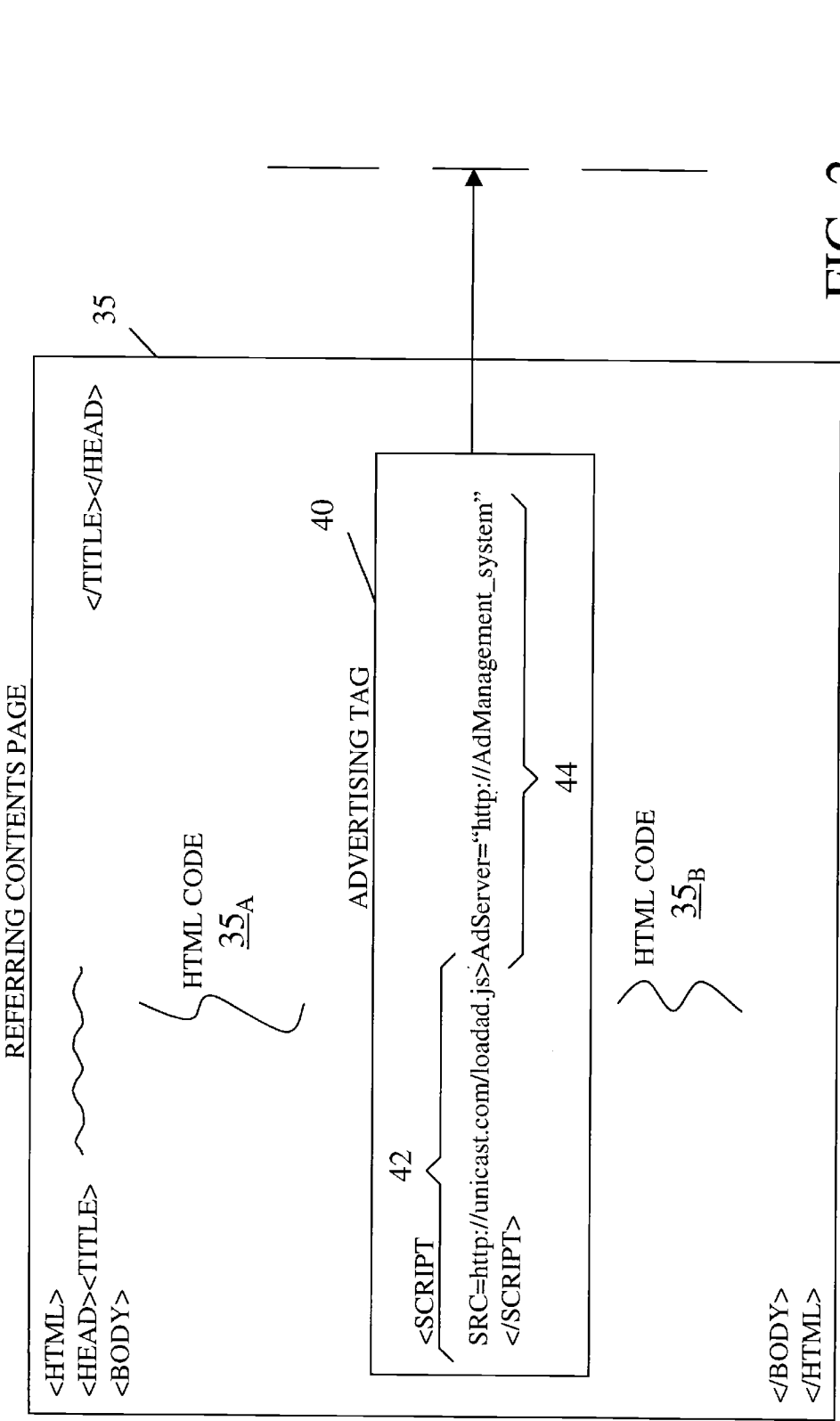


FIG. 2



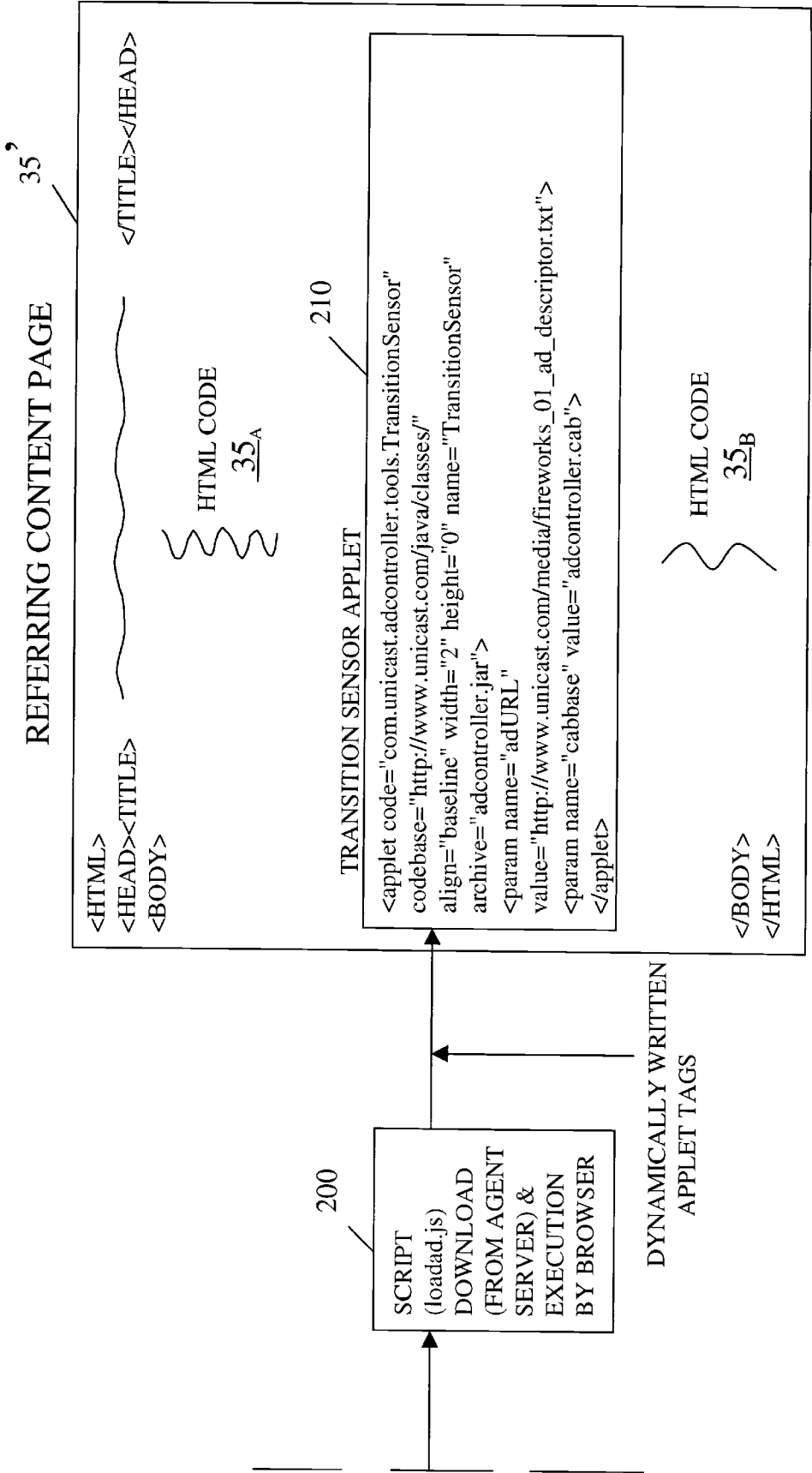
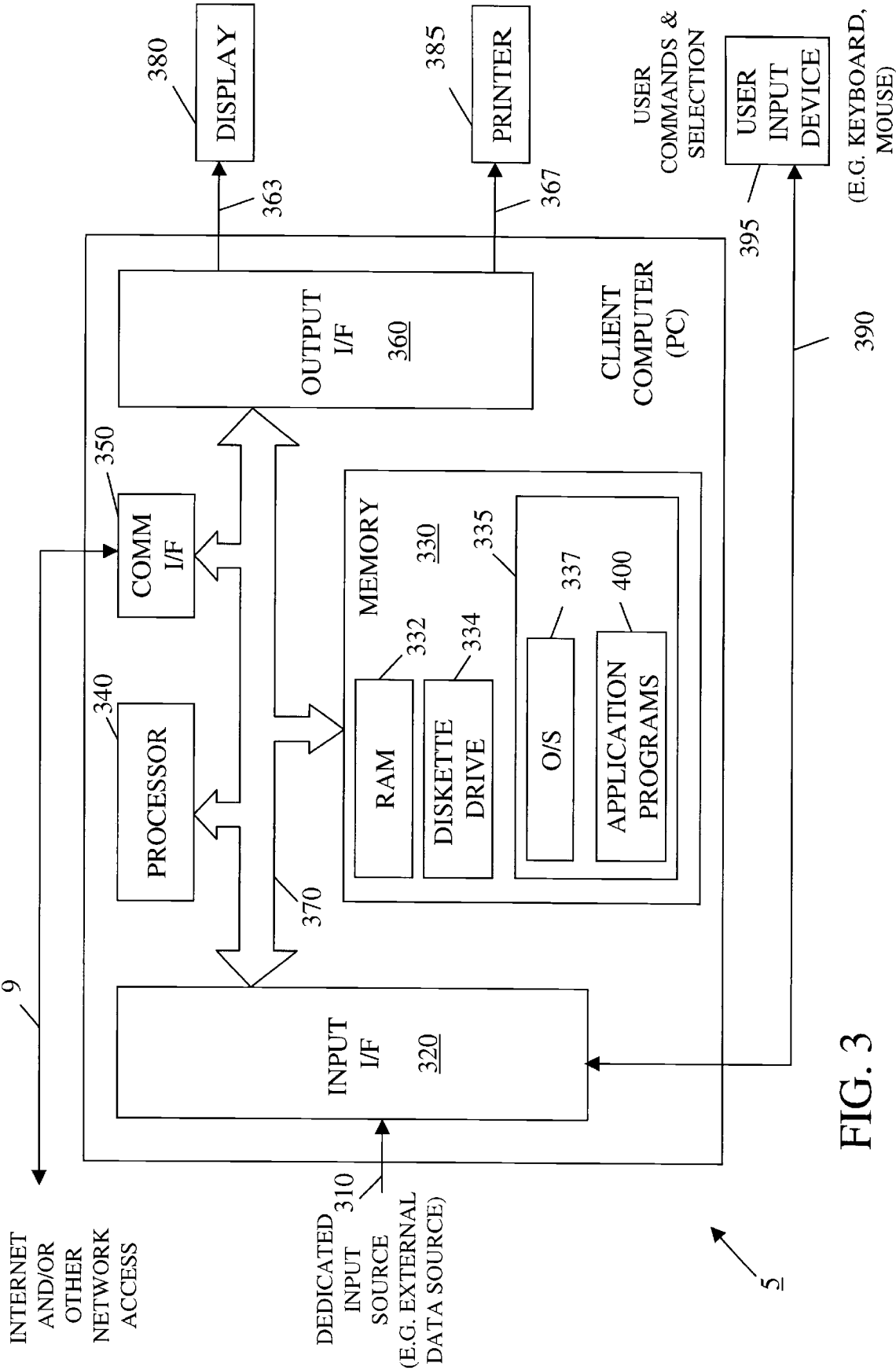


FIG. 2B



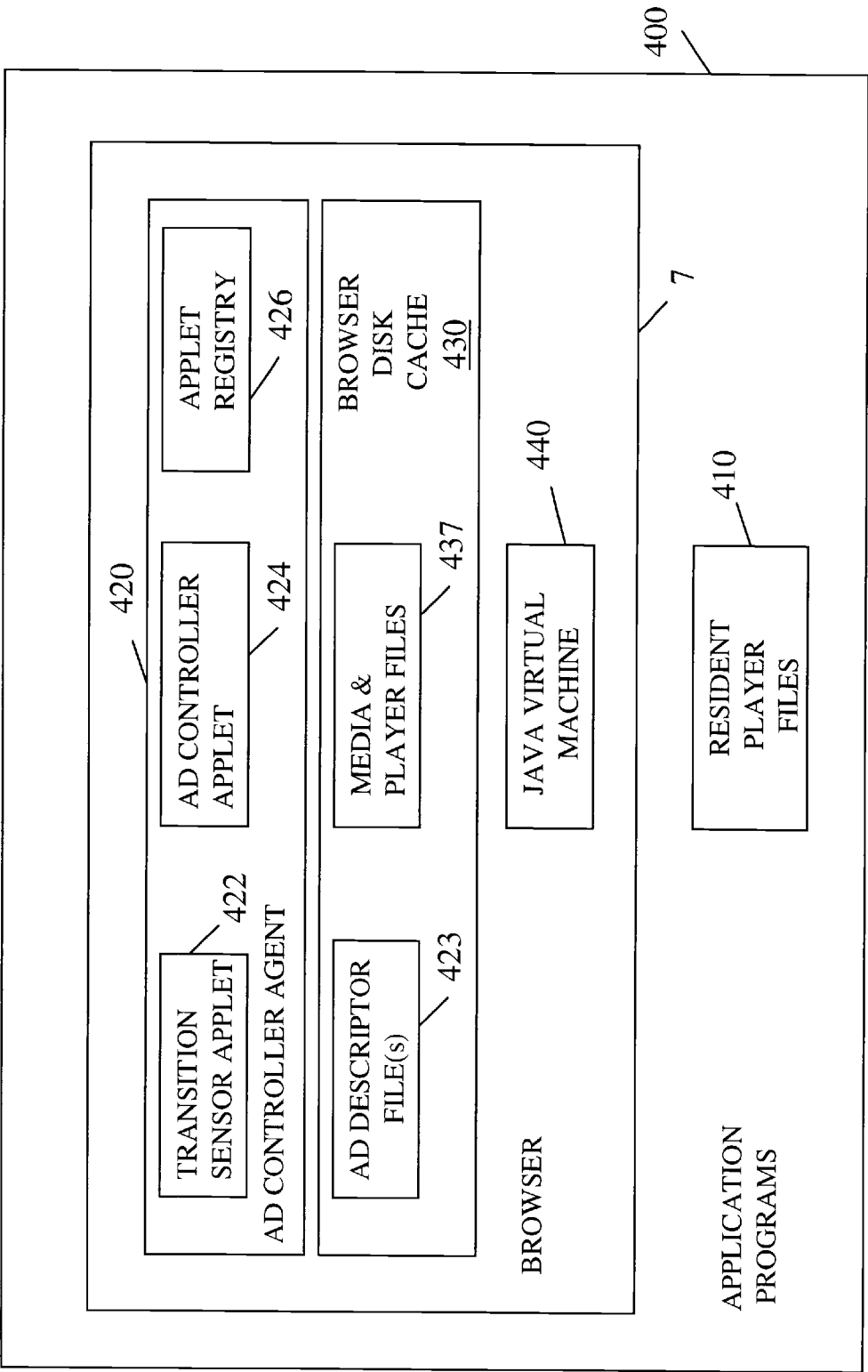


FIG. 4

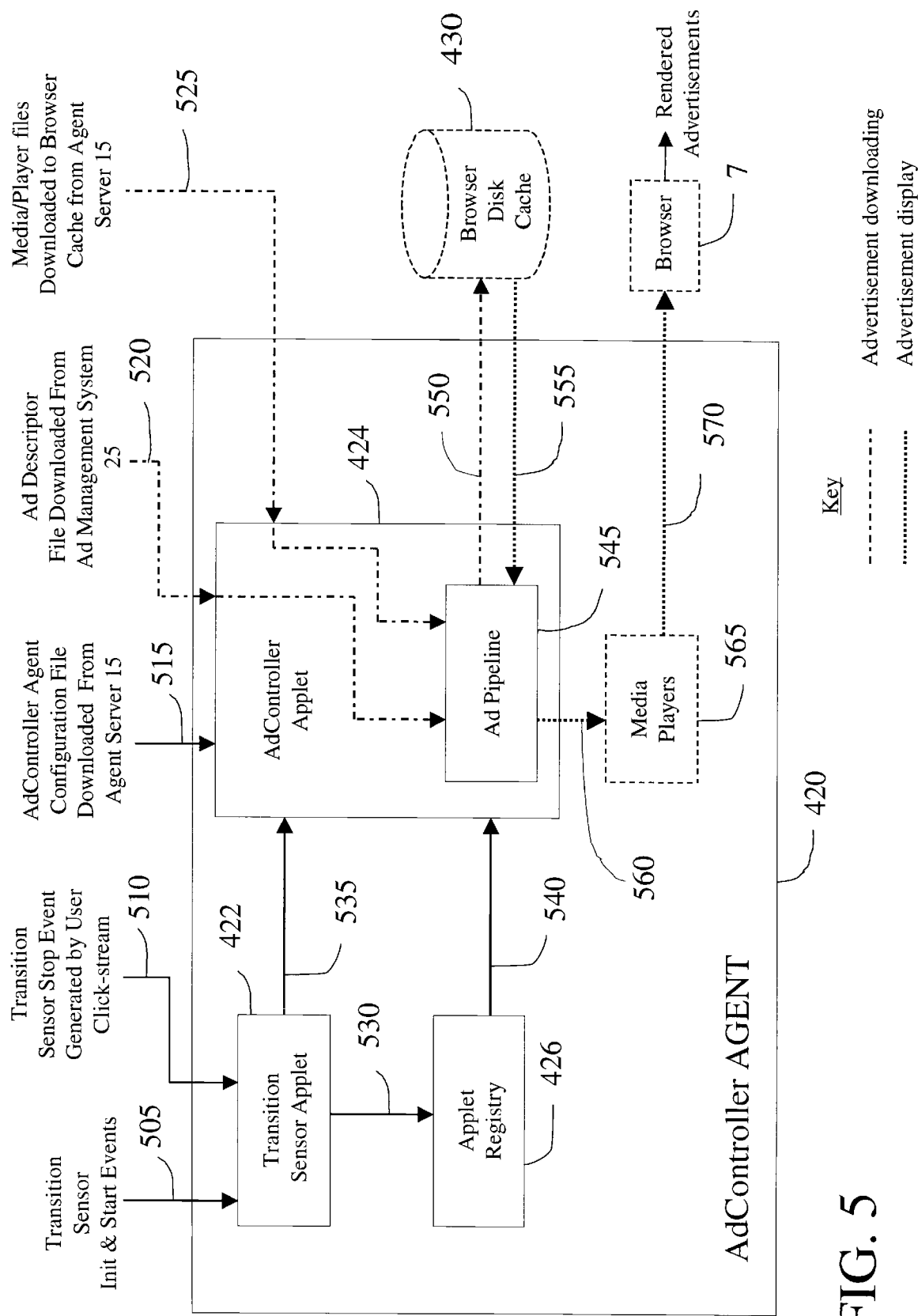
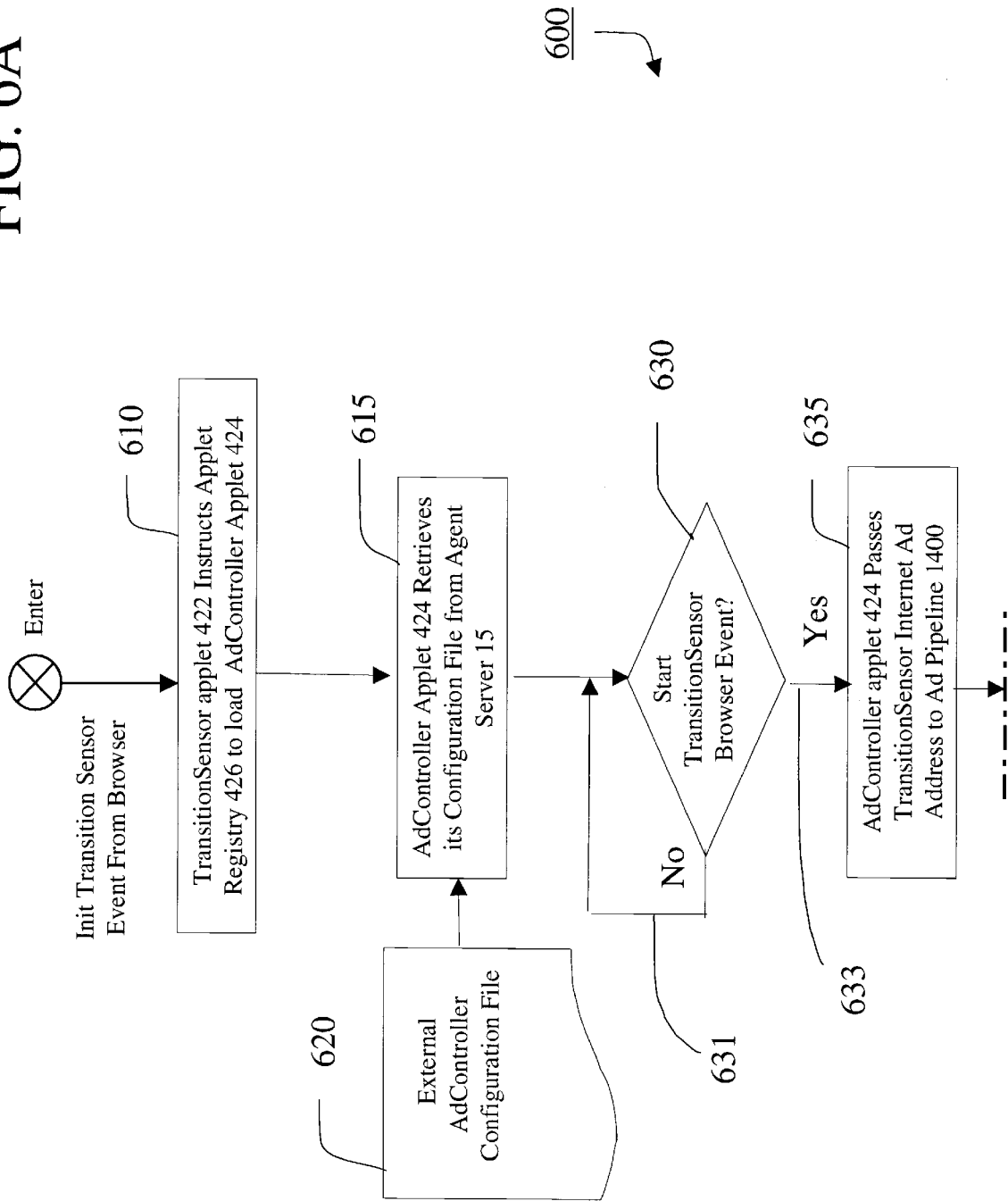


FIG. 5

FIG. 6A



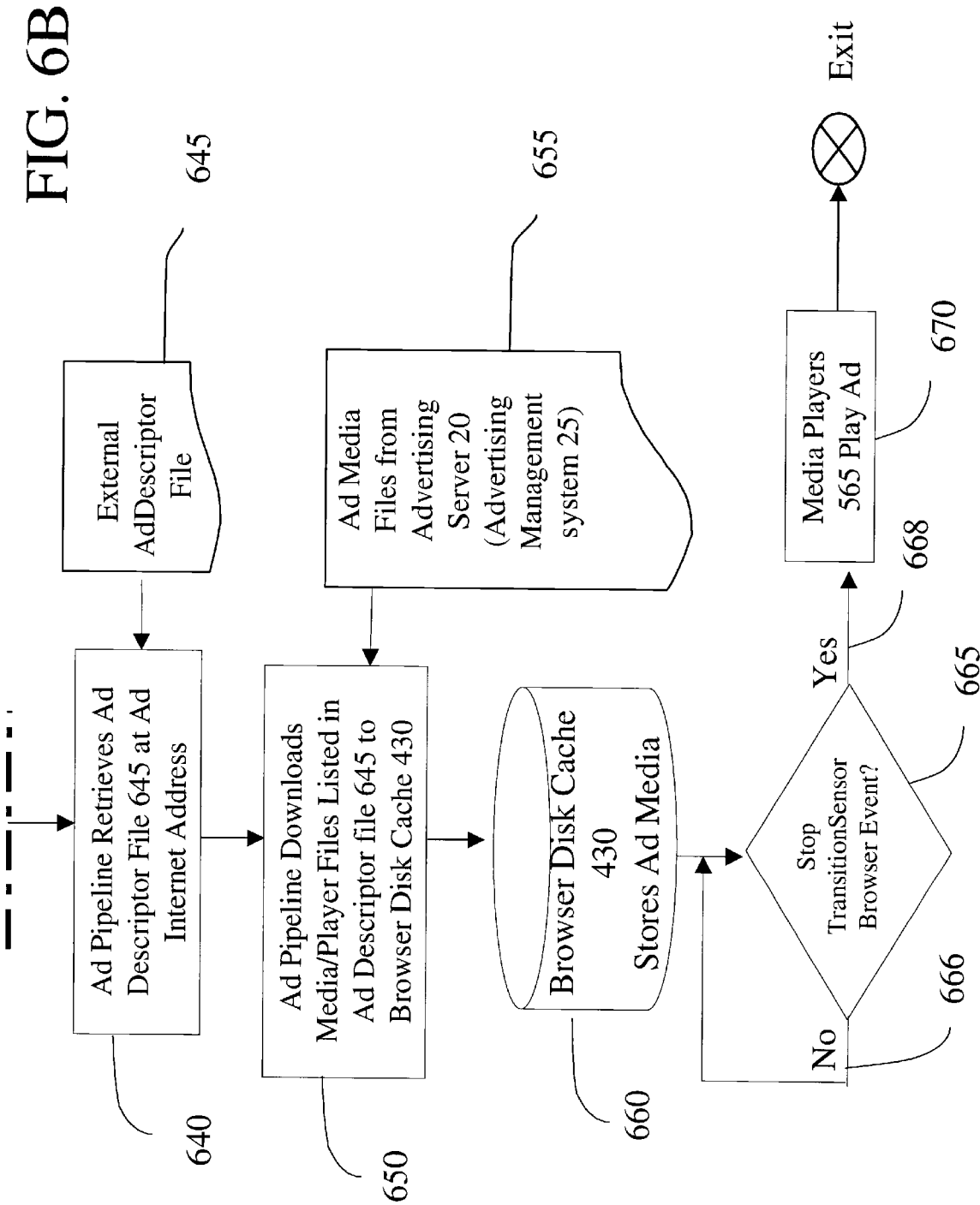


FIG. 6A
FIG. 6B

FIG. 6



FIG. 7

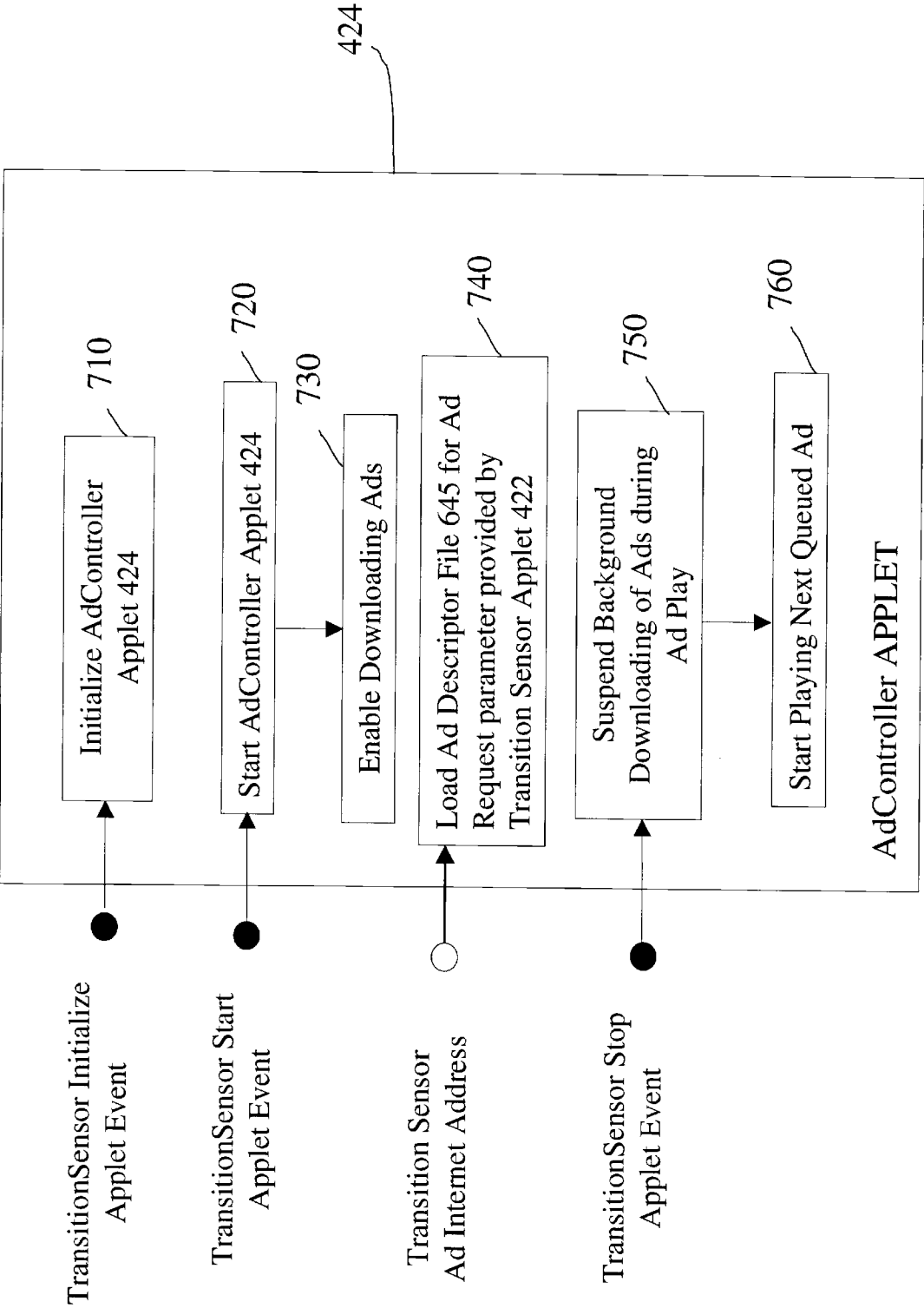


FIG. 8

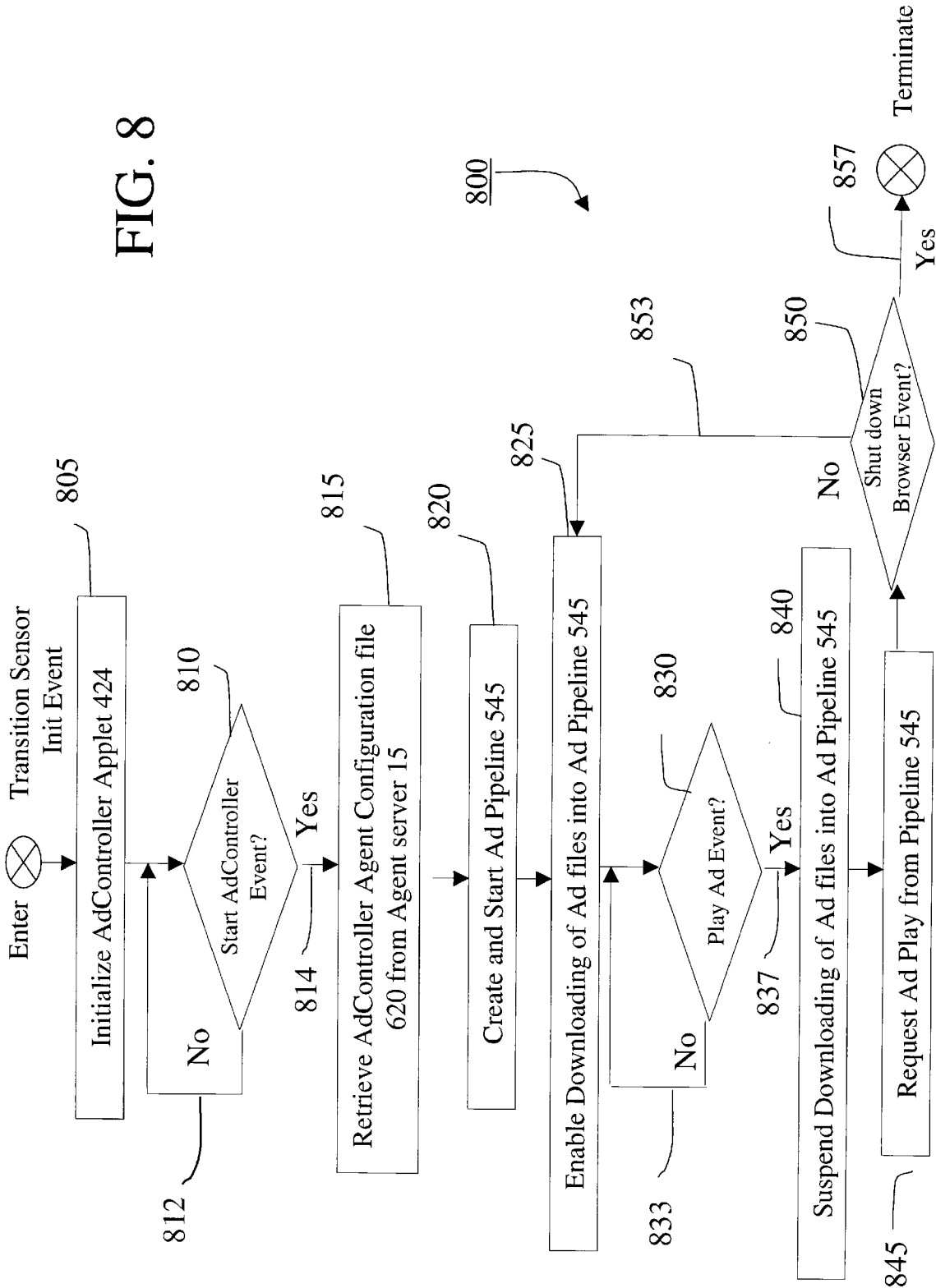
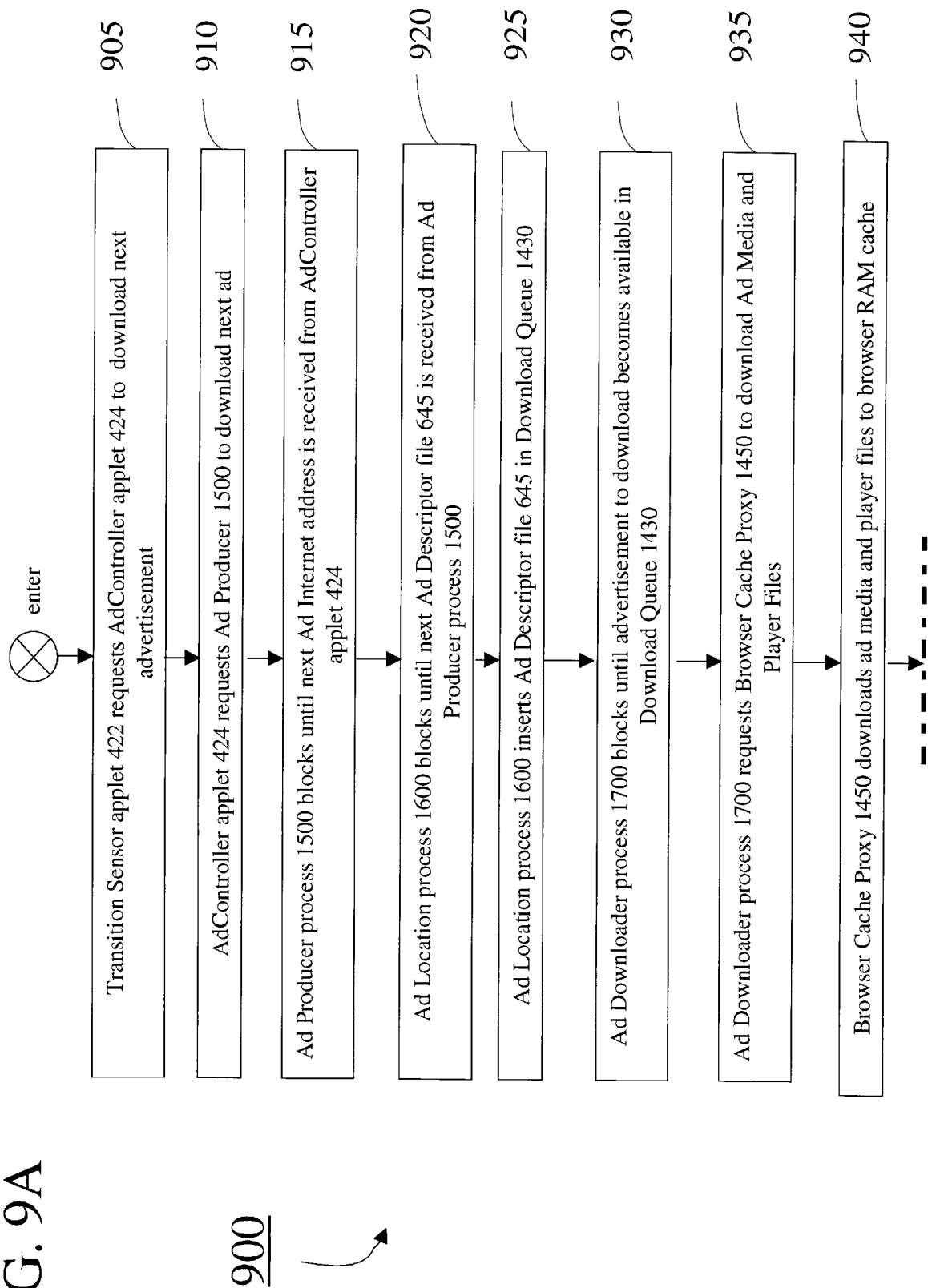


FIG. 9A



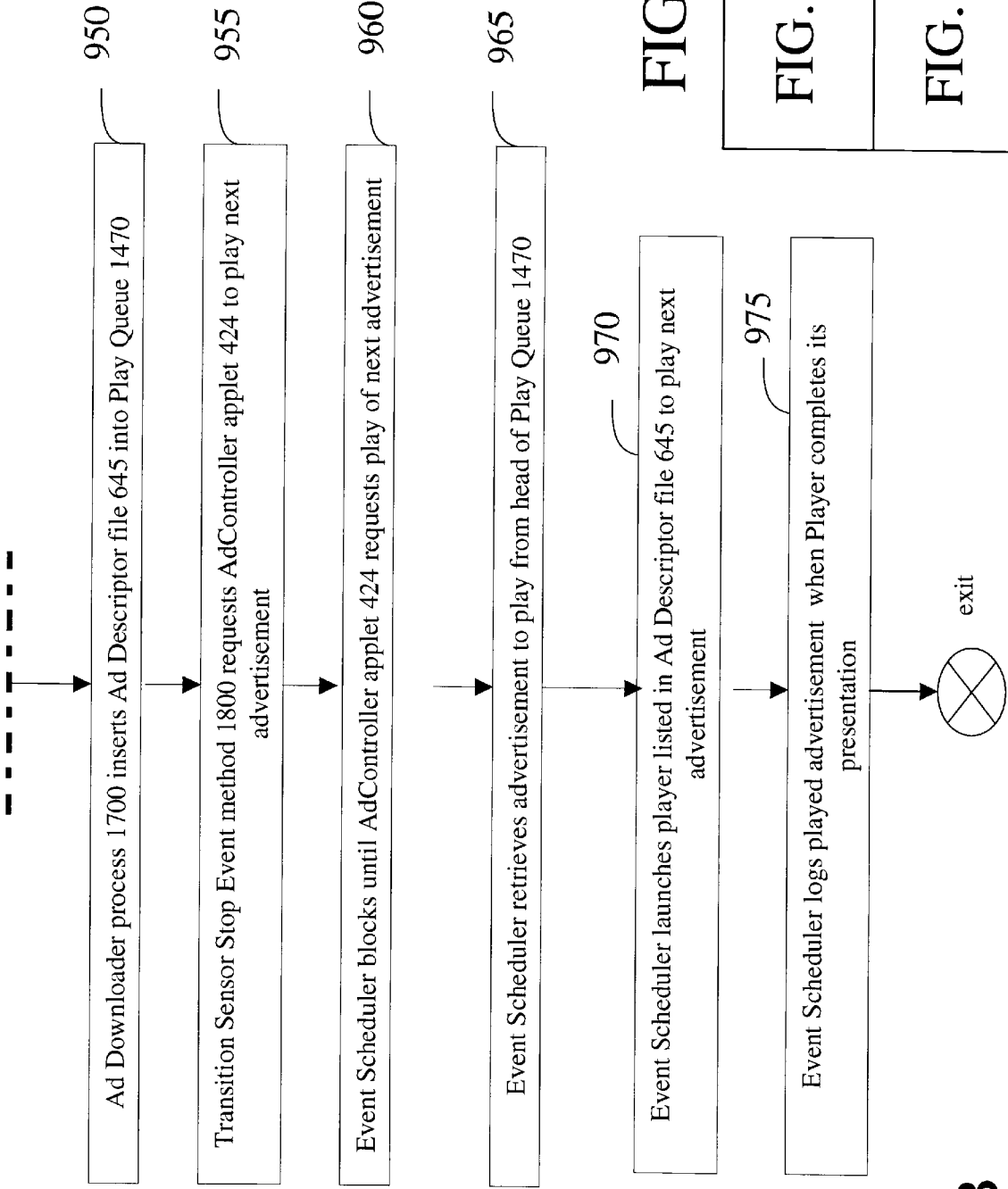


FIG. 9B

FIG. 9

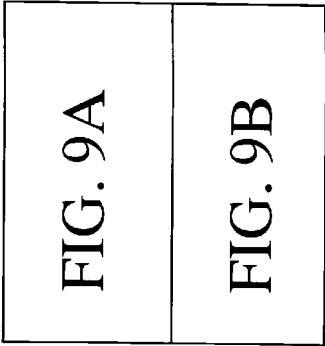
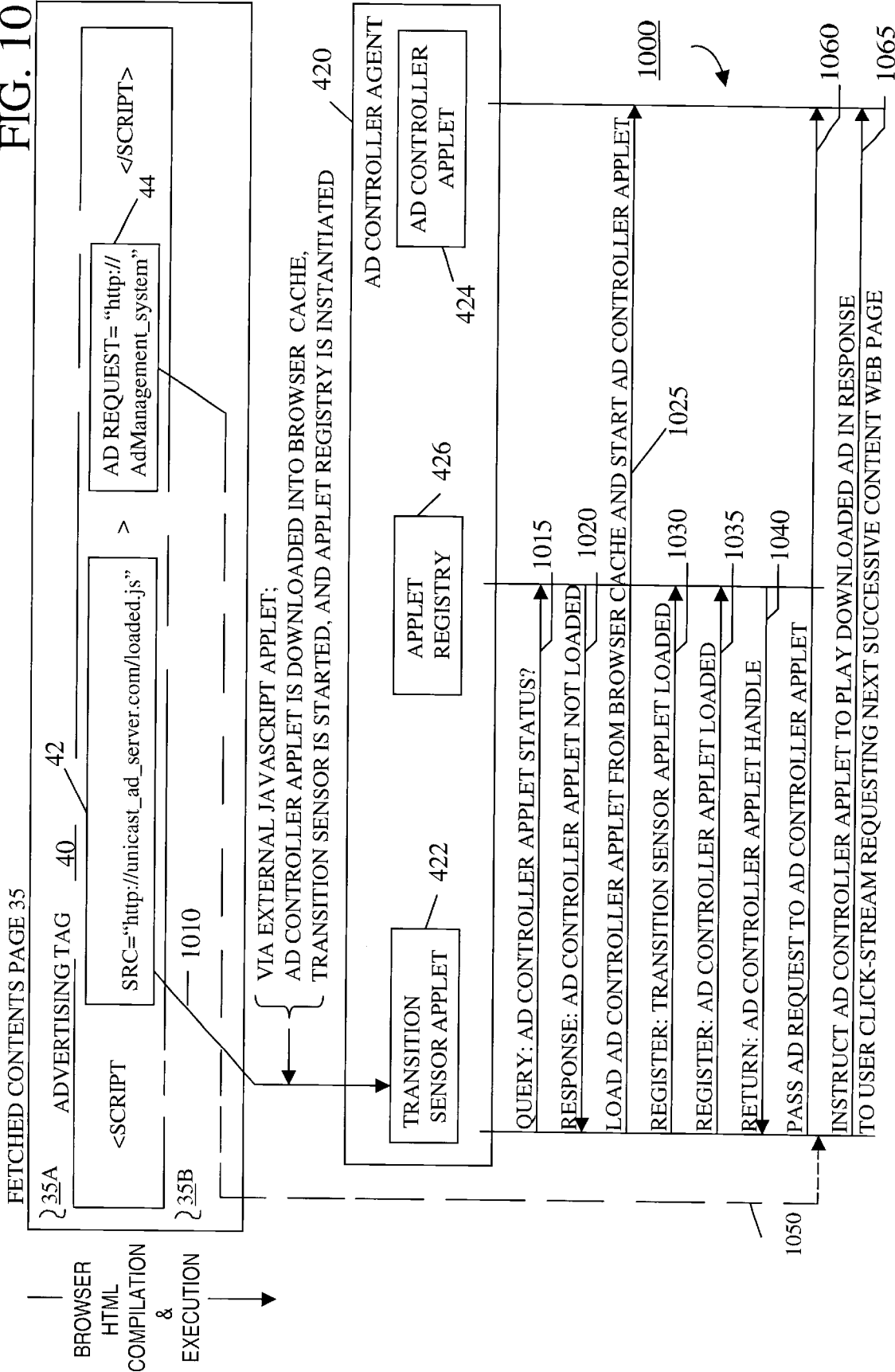


FIG. 10



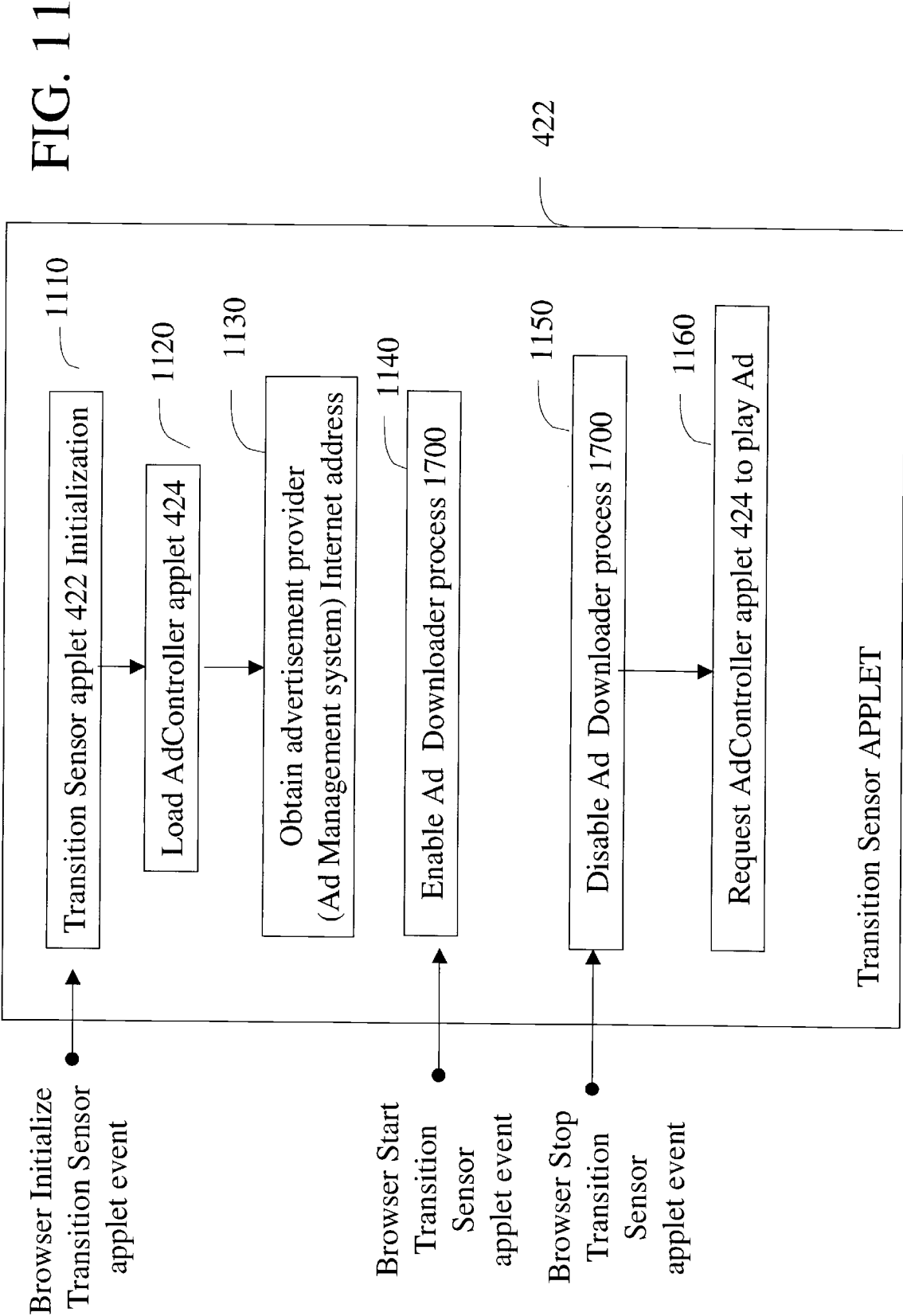
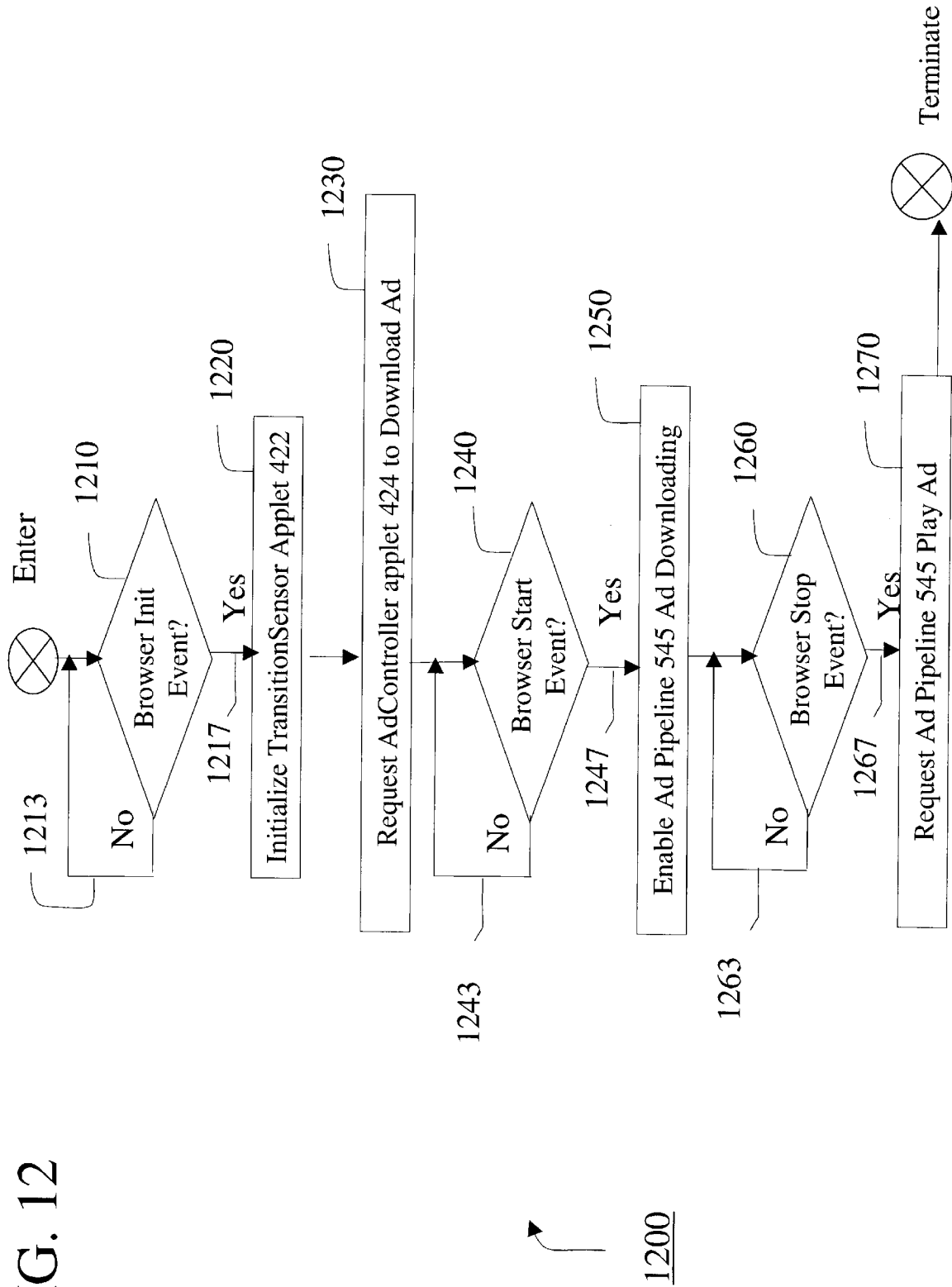


FIG. 12





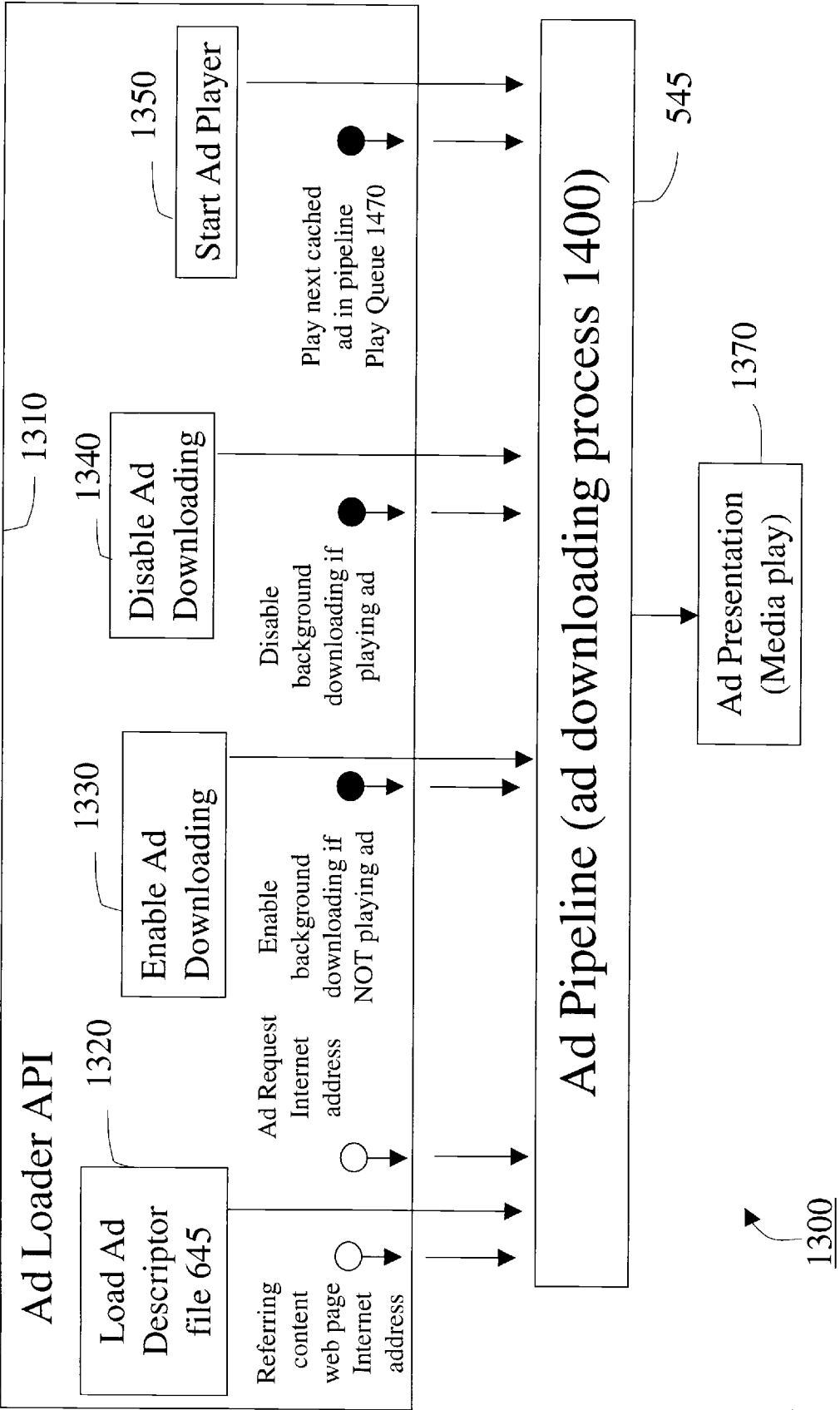
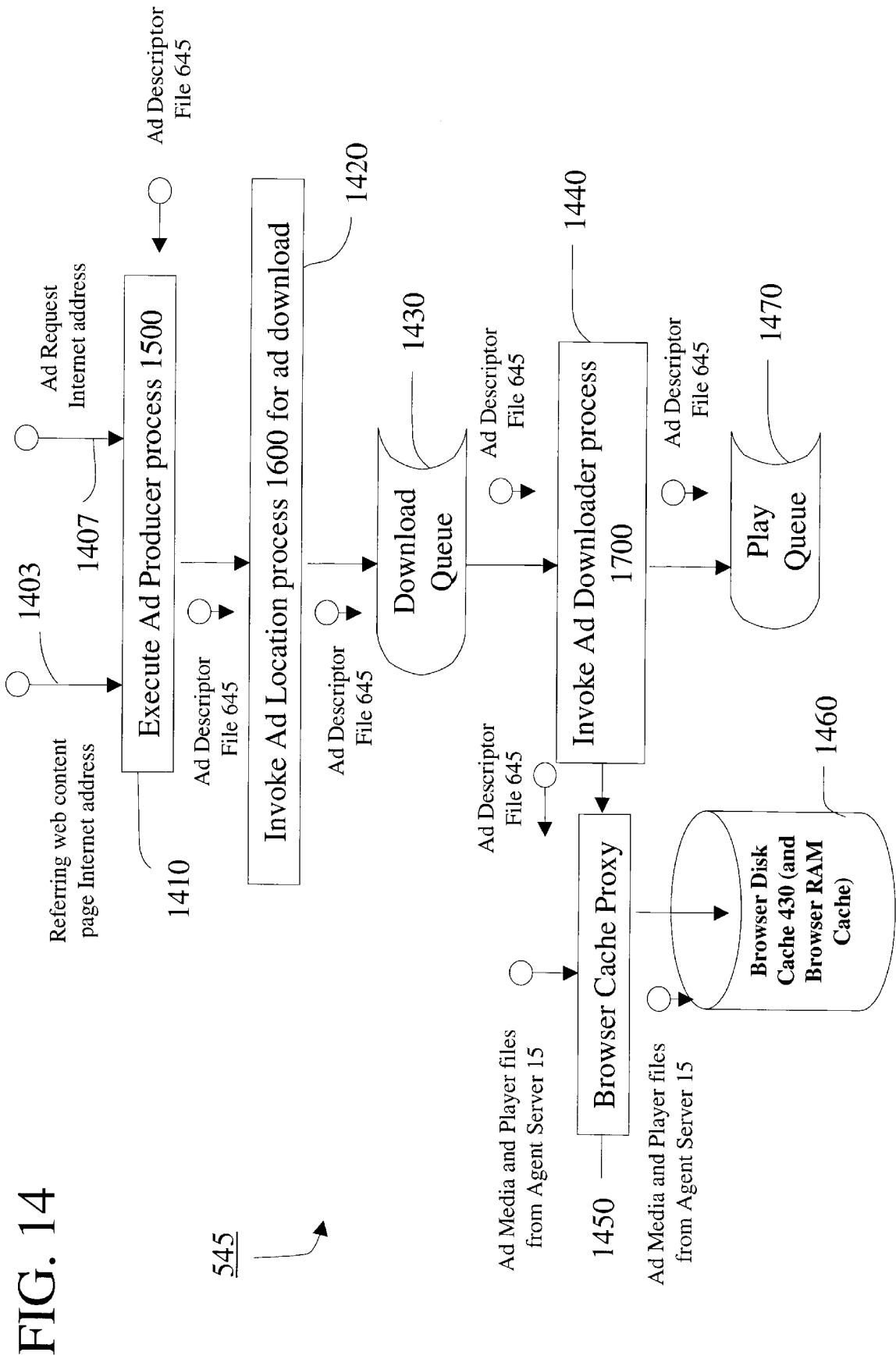


FIG. 13



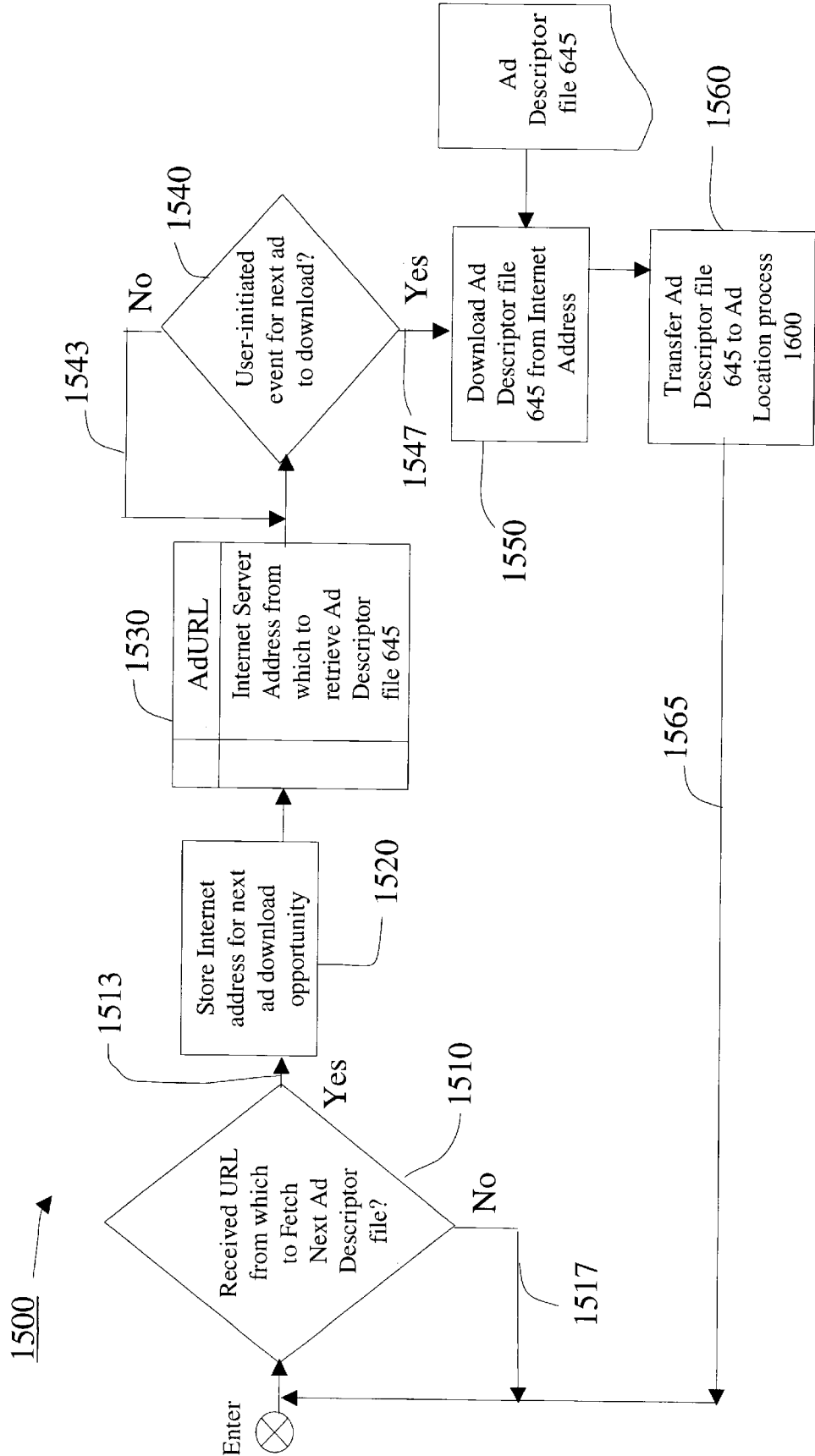
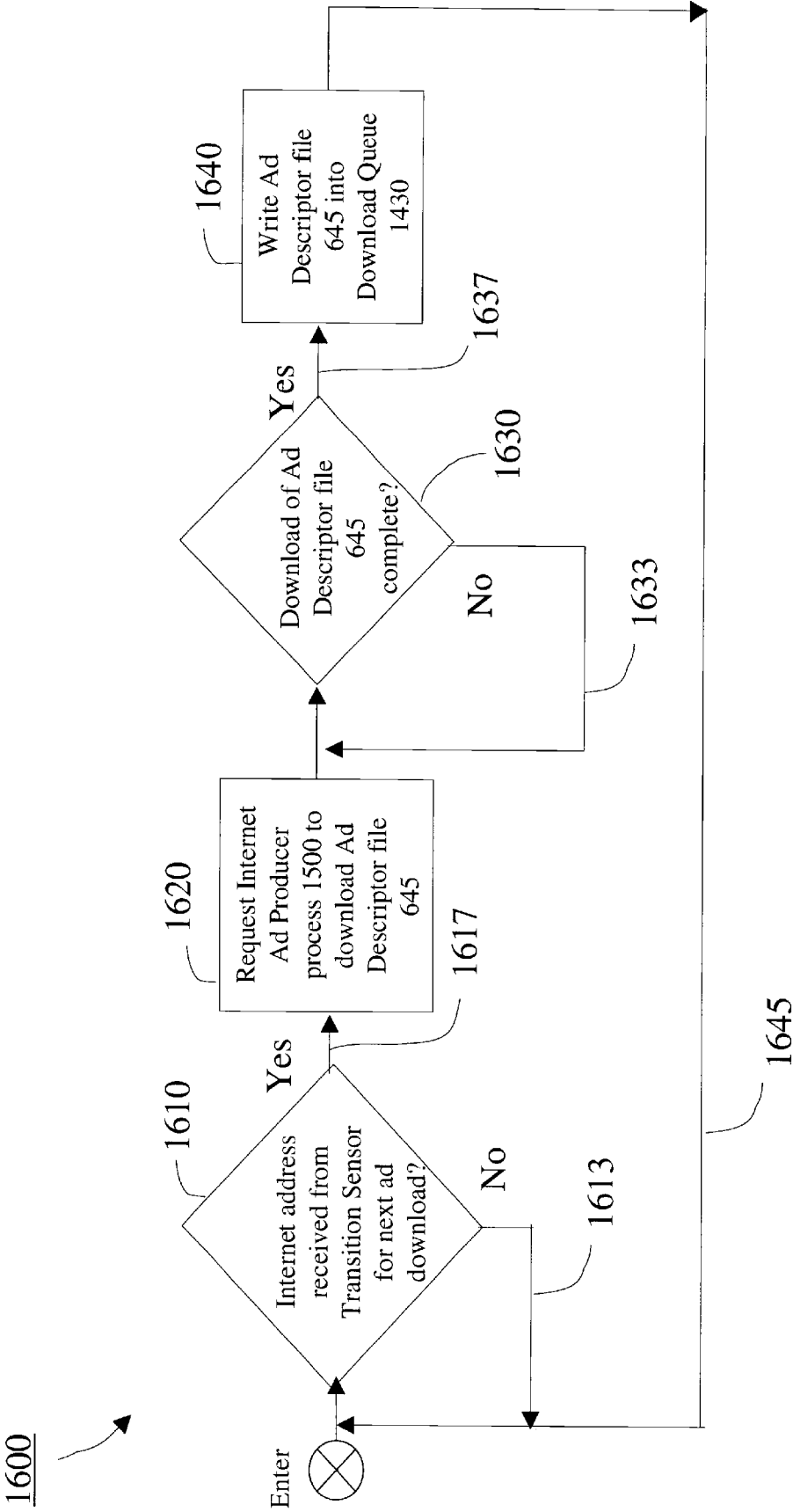


FIG. 15

FIG. 16



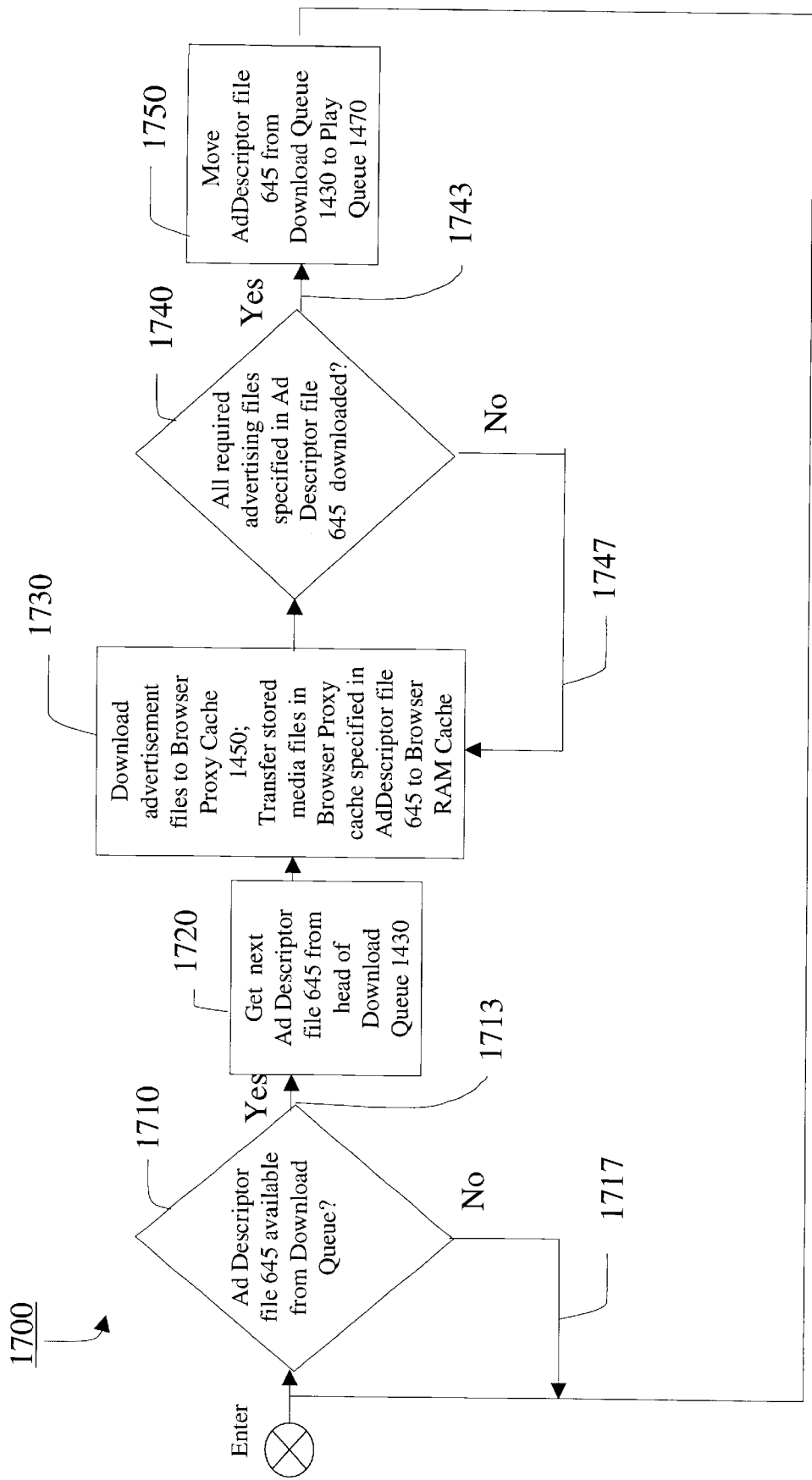


FIG. 17

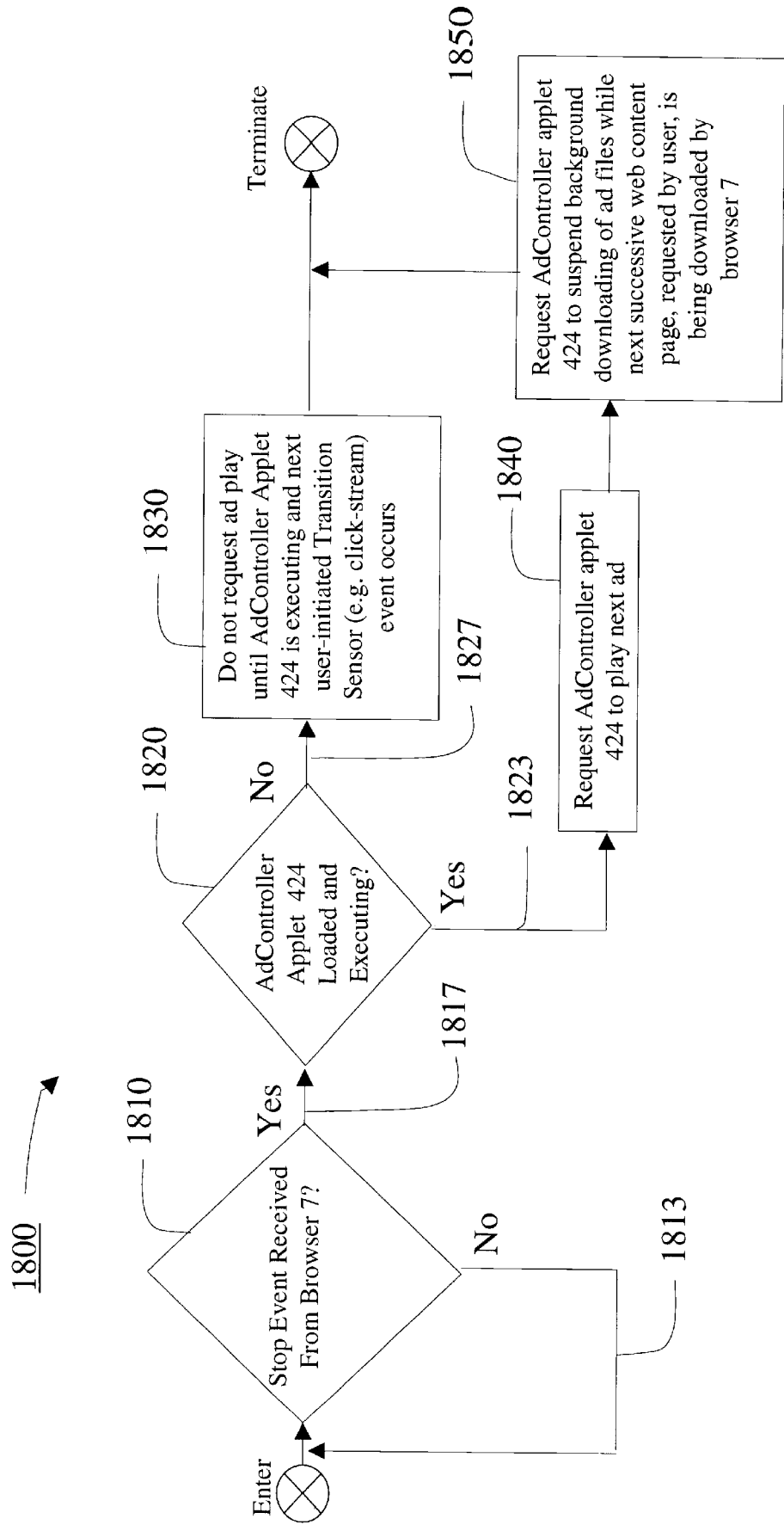


FIG. 18

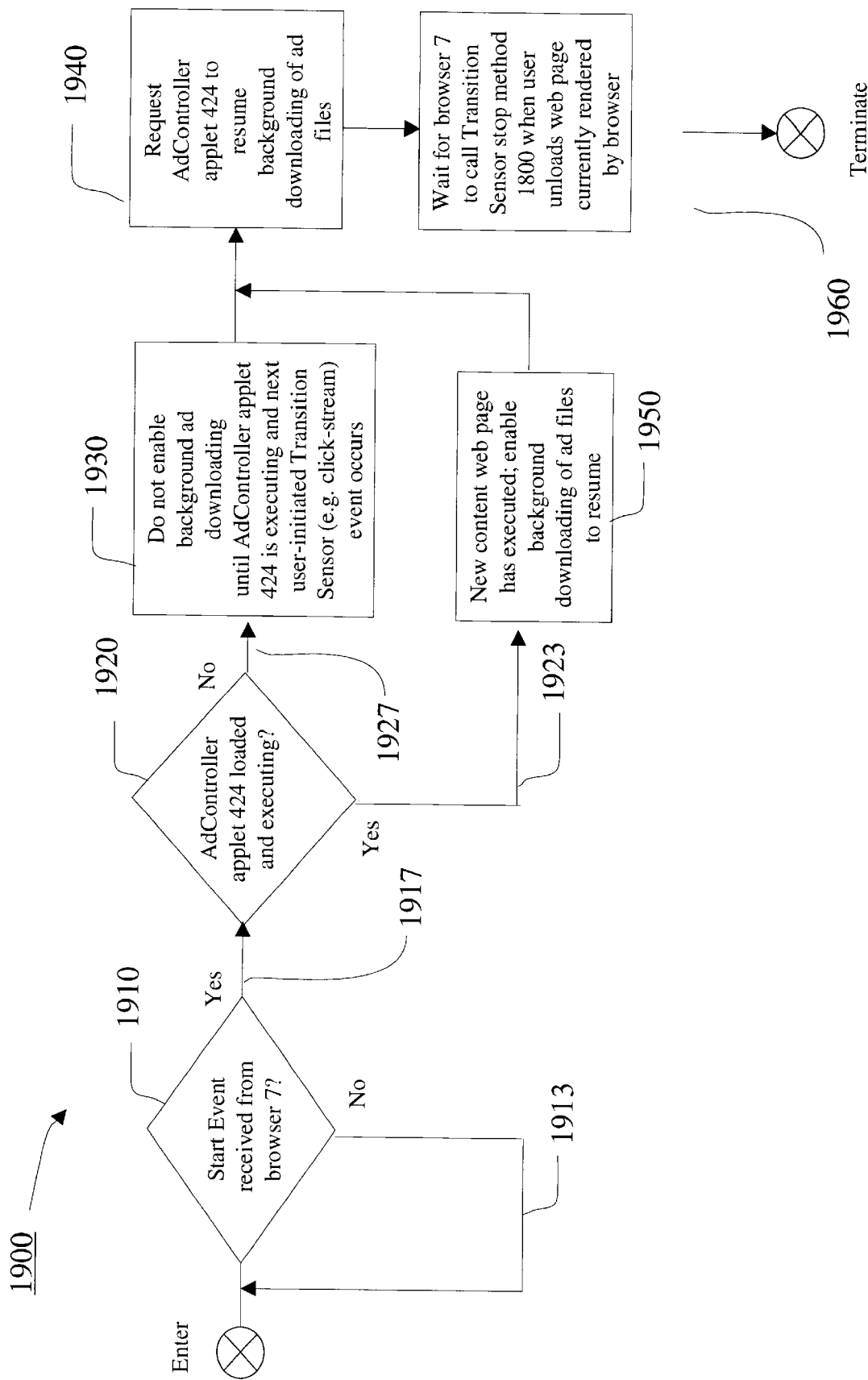


FIG. 19



**# Section 1 - Applet Player Java Class Configuration****playerName=AppletViewerPopup****playerClass=com.unicast.adcontroller.players.AppletViewerPopup****# Section 2 - Ad Java Classes Configuration****AppletViewerPopup.adAppletName=ANM\_AnimationLoaderApplet****AppletViewerPopup.adAppletClass=**

com.pointcast.applets.AnimationApplet.ANM\_AnimationLoaderApplet

**# Section 3 - Player Execution Configuration**

AppletViewerPopup.windowTitle=AdController PointCast Ad

AppletViewerPopup.playerRefreshRate=1000

AppletViewerPopup.allowExit=true

AppletViewerPopup.xPosition=50

AppletViewerPopup.yPosition=50

AppletViewerPopup.windowWidth=280

AppletViewerPopup.windowHeight=355

AppletViewerPopup.isResizable=false

AppletViewerPopup.secondsWindowsOpen=180

AppletViewerPopup.secondsToOverlay=1

AppletViewerPopup.closeButtonLabel=Close

AppletViewerPopup.openButtonLabel=More Info

AppletViewerPopup.saveButtonLabel=Save

AppletViewerPopup.openURL=http://www.pointcast.com/

**# Section 4 - Ad Applet Configuration****# A. Ad Applet DocumentBase**

AppletViewerPopup.ANM\_AnimationLoaderApplet.documentBase=

http://www2.unicast.com/~rlandsma/AdController/MacromediaApplet/

**# B. Ad Applet Parameters**

AppletViewerPopup.ANM\_AnimationLoaderApplet.AdToPlay=deepsea.anm

AppletViewerPopup.ANM\_AnimationLoaderApplet.AltImage=test.gif

AppletViewerPopup.ANM\_AnimationLoaderApplet.MaxCycles=5

AppletViewerPopup.ANM\_AnimationLoaderApplet.TargetURL=http://www.pointcast.com/

AppletViewerPopup.ANM\_AnimationLoaderApplet.TargetFrame=\_top

AppletViewerPopup.ANM\_AnimationLoaderApplet.BorderWidth=2

AppletViewerPopup.ANM\_AnimationLoaderApplet.BorderType=Standard

**# C. Ad Applet MediaList**

AppletViewerPopup.ANM\_AnimationLoaderApplet.mediaURLList=new

AppletViewerPopup.ANM\_AnimationLoaderApplet.mediaURLList.size=2

AppletViewerPopup.ANM\_AnimationLoaderApplet.mediaURLList.element0=deepsea.anm

AppletViewerPopup.ANM\_AnimationLoaderApplet.mediaURLList.element0=animApplet.jar

2000

FIG. 20

**TECHNIQUE FOR IMPLEMENTING  
BROWSER-INITIATED USER-  
TRANSPARENT ADVERTISING AND FOR  
INTERSTITIALLY DISPLAYING AN  
ADVERTISEMENT, SO DISTRIBUTED,  
THROUGH A WEB BROWSER IN RESPONSE  
TO A USER CLICK-STREAM**

**CROSS-REFERENCE TO RELATED  
APPLICATION**

This application is a division of co-pending patent application Ser. No. 09/237,718, filed Jan. 26, 1999 and entitled "A TECHNIQUE FOR IMPLEMENTING BROWSER-INITIATED USER-TRANSPARENT NETWORK-DISTRIBUTED ADVERTISING AND FOR INTERSTITIALLY DISPLAYING AN ADVERTISEMENT, SO DISTRIBUTED, THROUGH A WEB BROWSER IN RESPONSE TO A USER CLICK-STREAM", which itself is a continuation-in-part of, now abandoned, patent application Ser. No. 09/080,165, filed May 15, 1998 and entitled "LOCALLY-SUMMONED NETWORK-DISTRIBUTED CONFIRMED INFORMATIONAL PRESENTATIONS".

**BACKGROUND OF THE DISCLOSURE**

**1. Field of the Invention**

The invention relates to a technique, specifically apparatus and accompanying methods, for implementing in a networked client-server environment, such as the Internet, network-distributed advertising in which an advertisement is downloaded, from an advertising server to a web browser executing at a client computer, in a manner transparent to a user situated at the browser, and subsequently displayed, by that browser and on an interstitial basis, in response to a click-stream generated by the user to move from one web page to the next.

**2. Description of the Prior Art**

Currently, Internet usage, and particularly that of the World Wide Web (henceforth referred to as simply the "web"), is growing explosively, particularly as the number of web sites and users that have access to the Internet continue to rapidly and to a great extent, exponentially, expand.

In essence, after establishing a suitable network connection to the Internet, a user at a client computer can easily employ a graphical web browser, such as the Internet Explorer ("IE") browser presently available from Microsoft Corporation of Redmond, Wash., to connect to a web site and then download a desired web page by simply supplying a specific address (known as a URL or uniform resource locator) of that page to the browser. The URL identifies both an address of the site, in terms of its Internet domain name, and a page of information at that site, in terms of its corresponding file name. Each web site stores at least one, and often times substantially more pages all arranged in a pre-defined hierarchy, generally beginning, at its root, with a so-called "home page". Each such page is written in HTML (hypertext markup language) form. A page, in this context, refers to content accessed via a single URL, including, e.g., text, graphics and other information specified in the HTML code for that particular page. Once a user supplies a URL of interest, the browser operated by that user sends an appropriate command, using a TCP/IP protocol (transmission control protocol/internet protocol), to a remote HTTP (hypertext transport protocol) server, located at the web site and which stores that page, to access and download a corresponding file for that page. In response, the server

then sends, using the TCP/IP protocol, a stored file containing HTML code that constitutes that page back to the browser. As the file that constitutes the page itself is received by the browser, the browser interprets and executes the HTML code in that file to properly assemble and render the page on, e.g., a monitor to a user situated at the client computer. Such a page may itself contain HTML commands that reference other files, residing on the same or different web sites, which, when these commands are appropriately interpreted and executed by the browser, result in those files being downloaded and their resulting content properly assembled by the browser into the rendered page. Once all the content associated with the page is rendered, the user can then position his(her) mouse cursor on a suitable hypertext link, button or other suitable user input field (whichever here implements a "hotlink") displayed on that page and then, through, e.g., a mouse "click", effectively download a file for and render another desired page in succession until the user has finished his(her) visit to that site, at which point, the user can transition through a hotlink to a page at another site, and so forth. A hotlink specifies a complete web address of an associated page, including a domain name of its hosting web site at which that page is situated. Consequently, by simply and successively positioning and "clicking" his(her) mouse at an appropriate hotlink for each one of a number of desired web pages, the user can readily retrieve an HTML file for each desired page in succession from its corresponding web site and render that page, and, by doing so, essentially effortlessly jump from site to site, regardless of where those sites are physically located.

Ever since their introduction several years ago, HTML and accompanying browser software, now including, e.g., attendant programming languages such as Java and JavaScript languages ("Java" is a registered trademark of Sun Microsystems in Mountain View, Calif.; "JavaScript" is a trademark of Netscape Communications in Mountain View, Calif.), have undergone rather rapid and continual evolution. A major purpose of which has been and continues to be to provide web page authors with an ability to render increasingly rich content through their pages and, as a result, heighten a "user experience" for those users who visit these pages. Consequently, web pages are no longer limited to relatively simple textual displays—as occurred with early versions of HTML and browser software, but can now encompass even full-motion multimedia presentations and interactive games that use rather sophisticated graphics.

The simplicity of browsing the web coupled with the relative low-cost of accessing the Internet, and the relative ease through which a web site can be established are collectively fueling unparalleled growth and diffusion of the Internet itself, web sites and the Internet user community throughout the world. In that regard, by establishing web sites, merchants, vendors and other information providers have an unparalleled opportunity, basically unheard of as little as 5–10 years ago, to reach enormous numbers of potential consumers—regardless of where these consumers reside—at costs far less than previously thought possible. Moreover, given the staggering amount and wide diversity of information currently available on the web, web browsing is becoming so popular a past-time for sufficient numbers of individuals that browsing is beginning to divert significant viewership away from traditional forms of mass entertainment, such as television and cable. While such diversion is relatively small at present, it is likely to rapidly grow. Moreover, given the ease and convenience with which users, situated at their personal computers and with basically nothing more complicated than a few mouse clicks, can

US 6,317,761 B1

3

effectively interact with remote web sites, electronic commerce, through which goods and services are ordered through the Internet without ever visiting a physical store, is rapidly emerging as a significant sales medium. This medium is likely to significantly challenge and possibly, over a relatively short time, may even alter traditional forms of retailing.

Given the wide and ever-growing reach of the web as a source of consumer information and the increasing consumer acceptance of electronic commerce, advertisers have clearly recognized the immense potential of the web as an effective medium for disseminating advertisements to a consuming public.

Unfortunately, conventional web-based advertising, for various practical reasons—some being technical in nature and others relating to a nature of traditional web advertisements themselves, has generally yielded unsatisfactory results and thus has usually been shunned by most large advertisers. In that regard, several approaches exist in the art for implementing web based advertisements. However, all suffer serious limitations of one form or another that have sharply restricted their desirability and use.

Currently, a predominant format, referred to as a “banner”, for a web advertisement takes the form of a rectangular graphical display situated, typically at a fixed location, in a rendered web page. A banner, which can be static or animated, can be situated anywhere within a rendered web page but most often is situated at a top or bottom, or along a vertical edge of that page. A banner, depending on its size, can extend across an entire page width or length, and usually contains, in a graphical eye-catching form, a name of a product or service being advertised. Increasingly, a banner for a given product or service implements a hotlink to enable a consumer to “click-through” the banner (i.e., generate a mouse click on the banner) in order to transition, via his browser, to a web site maintained by a corresponding advertiser and, from that site, fetch a web page to provide additional information regarding that product or service. Hence, the consumer could easily obtain more information by a click-through; while an advertiser, monitoring counts of such click-throughs that occur in a given period of time, could gain feedback on the effectiveness of the corresponding banner.

A banner is generally produced by properly embedding specific HTML code for that banner within the HTML coding for a given web page in which the banner is to appear. A client browser, as it interprets and sequentially executes the HTML code for a fetched page, will, in turn, compile and execute the embedded code for the banner and hence display the banner, as part of a rendered page and at a specified location thereon.

In implementing a banner, whether static or even animated, its HTML coding generally involved downloading an appropriate file, for that banner, to a client browser. The file may be stored on the same server that stores the HTML file for the page, or accessed from a remote server. The file may contain a graphic itself, such as in a GIF (graphic interchange format) file, or a Java applet which, once interpreted and executed by the browser, generates and renders a desired animated graphic. This file, whether it be a graphic or applet, requires time to download and must be downloaded and assembled by the browser on the page prior to that page being fully rendered. The download time for that file, particularly as it increases in size, clearly, a priori, lengthens a time interval during which that page would completely download, thereby extending the time to fully

4

render the page, including the banner, after a user transitioned to that page. Channel bandwidth to a client computer (e.g., personal computer—PC), such as that provided through a modem connection, is often rather limited. Consequently, if the file size for the banner were relatively large—as would certainly be the case for relatively “rich” content, e.g., audio or video content, the delay in downloading such a file over such a limited bandwidth connection could be excessive, and consequently highly frustrating to the user. Hence, a user would likely wait a considerable amount of time before all the page components for multimedia content are fully downloaded to permit that page to be rendered. Such delay, if encountered during a page transition, can be rather frustrating to a user, even to the point at which the user, just to end his(her) waiting, will prematurely terminate the download and transition to another page. Therefore, in an effort to preserve an appropriate “editorial experience” for a user, content suppliers sharply limit the file size, of such banners to be rendered on their pages, in order to minimize page download and hence latency times.

Unfortunately, such restricted file sizes effectively limit the richness of the content of a banner to a rather simplistic advertisement—even with animation. Thus, banners often failed, as advertisers soon recognized by relatively low click-through counts, to attract sufficient viewer attention to justify their use and expense.

In an effort to overcome the content limitation associated with banners, the art teaches the use of a different advertising modality: so-called “interstitial” advertisements. See, e.g., U.S. Pat. No. 5,305,195 (issued to A. J. Murphy on Apr. 19, 1994—hereinafter the “Murphy” patent) which discloses the concept of using interstitial advertisements though not in the context of web advertising. As described in the Murphy patent, pre-stored advertisements are displayed at specific intervals on each one of a group of networked ATM (automated transaction machines) terminals. In particular, the advertisements are downloaded, either directly or via a server, from a remote computer and locally stored on each such terminal and subsequently displayed on that terminal while it waits for a response, from a remote mainframe transaction server, to a transaction initiated at that terminal.

Generally speaking and with specific reference to web advertising, interstitial ads are displayed in an interval of time that occurs after a user has clicked on a hot-link displayed by a browser to retrieve a desired web page but before that browser has started rendering that page. Such an interval, commonly referred to as an “interstitial”, arises for the simple reason that a browser requires time, once a user clicks on a hotlink for a new page, to fetch a file(s) from a remote web server(s) for that particular page and then fully assemble and render that page. The length of an interstitial interval, which is quite variable, is governed by a variety of factors, including, e.g., a number of files required to fully render the new page and the size of each such file, and network and server congestion and attendant delays occurring when the user activated the hotlink.

Interstitial web advertising is taught in, e.g., U.S. Pats. 5,737,619 and 5,572,643 (both of which issued to D. H. Judson but on Apr. 7, 1998 and Nov. 5, 1996, respectively—hereinafter the “Judson” patents). The Judson patents disclose the concept of embedding an advertisement, as an information object, in a web page file in such a manner that the object will remain hidden and not displayed when the file is executed to render the page. Rather than being displayed, the information object is locally cached by the browser during execution of the code for that page. Then, during a

US 6,317,761 B1

5

transition initiated by user activation of a hotlink to move from that page to a next successive page, i.e., during an interstitial, the browser accesses the advertisement from local cache and displays it until such time as that next successive page is downloaded and rendered. See also, published International patent application WO 97/07656 (to E. Barkat et al and published on Mar. 6, 1997) which teaches the concept of “polite” downloading. Here, a browser, on a local computer (e.g., a client PC) downloads, from a remote advertising system server and ostensibly as a background process, file(s) for a web advertisement only during those intervals when bandwidth utilization of a communication channel (link) connected to the browser is less than a pre-established threshold. Such “polite” downloading is intended to minimally interfere with other communication applications, then executing on the client PC, which will utilize the link. The browser displays the downloaded ad(s) to the user only after the user has not interacted, as detected by a conventional screen saver process, with his(her) PC for a predefined period of time, such as by neither moving a mouse nor depressing a key on a keyboard during that period. The server selects those advertisements for download to the client PC based on a user-ID and preference information of the user, who is then situated at that PC, and configuration information of that PC, which, when a connection is established between the client PC and the server, the client PC uploads to the server. Though the files associated with an interstitial advertisement can be large, these files are advantageously fetched by a client browser during those intervals when otherwise the browser would be idle and hence bandwidth utilization of its network connection would be relatively low. Such “idle times” would occur, in the absence of processing an interstitial ad, after the browser has fully rendered a web page and a user is viewing the page but has not yet clicked on a hotlink to transition to another page. During such an idle time, the browser would simply wait for further user input.

By reducing, if not eliminating, problems, inherent in banners and engendered by download latency, interstitial web advertisements, by employing idle time downloading and local caching, provide a theoretical promise of conveying very rich media content with a pleasing “user experience”. However, interstitial advertisements, as conventionally implemented, have serious practical deficiencies which have severely limited their use.

Conventional interstitial, as well as other forms of current, web advertisements—here not unlike banners—rely on embedding HTML ad code, as, e.g., a separate non-displayable object, within HTML coding for a web page. Unfortunately, this approach, inherent in that taught by the Judson patents, can be inflexible and expensive for an advertiser to implement and particularly later should that advertiser, for whatever reason, seek to modify his(her) ad content. In particular and presently, ad coding is manually inserted into each and every content web page that is to carry advertising. Consequently, insertion of increasingly sophisticated embedded advertising, such as multi-media or video or audio, in existing web site content requires a large investment in terms of human resources, time and cost as web sites, particularly large sites, increase a number of content pages available for advertising. In that regard, where a banner usually required insertion of, e.g., a single line of HTML code, content rich advertisements, such as those now implemented by parameterized embedded Java advertising applets, often consist of an entire page of coding and hence require far more extensive and increasingly labor-intensive and costly insertions. Moreover, over time, advertisers do

6

change their ads such as by replacing one ad with a totally new version. However, once HTML ad coding is embedded within a number of web pages, it can be quite impractical and rather costly for an advertiser to access each and every page in which his(her) ad coding has been inserted and then manually change the ad coding, as desired. The impracticality and attendant cost compound if these pages are copied to other web sites and hence diffuse through the Internet.

Given these deficiencies, the art teaches a concept of implementing web advertising through using so-called “push” technology. See, e.g., U.S. Pat. No. 5,740,549 (issued to J. P. Reilly et al on Apr. 14, 1998—hereinafter the “Reilly et al” patent). In essence and as described in the Reilly et al patent, a client PC, through execution of a “push” application program (called “administration manager”), establishes a network connection with an information server, i.e., a “push” web server, typically during off-hours, such as in the late evening or early morning, or at a predefined interval (e.g., every four hours). The information server then downloads, i.e., “pushes”, to the administration manager, content files, such as for advertisements and/or other predefined information, that are to be played to the user sometime later. The administration manager, i.e., the “push” application, in turn, stores all the “pushed” content files into a local database (referred to as the “information database”) on a local hard disk and, in response to instructions received from the information server, deletes those previously “pushed” content files which have already been displayed. The administration manager also maintains a user profile, which specifies user preferences as to the specific advertising and/or other information (s)he wants to receive, in the information database. As such, through each connection, the information server, by selecting content from its database relative to preferences specified in the user profile, attempts to “push” fresh content to the client PC that is likely to be of interest to the user but without duplicating that which was already displayed. Stored “pushed” content is later displayed, using a data viewer, either on user demand or during those times when the user is not interacting with the system, here too detected by a conventional screen saver procedure.

While push technology reduces download latency, by shifting downloads to occur at off-hours, this technology also suffers serious drawbacks which have greatly restricted its practical acceptance.

In particular, to access “pushed” content, a user must initially download and install to his(her) client PC a separate, platform-specific, software application program, as well as subsequent updates to that program as new push capabilities are released by the manufacturer of the program. Unfortunately, these application programs can often extend to tens of megabytes in length. Since typical Internet users establish modem connections to their Internet service providers, these users will find that downloading these relatively large program files, even in compressed form, will consume an inordinate amount of time and is generally impractical while (s)he is actively using his(her) client PC. Consequently, these users are constrained to purchasing, at some cost, an off-the-shelf version of the application program or downloading that program, typically at no cost for the program itself, at off-hours, when network congestion is relatively light. Furthermore, while some efforts are underway in the art to automatically “push” and install incremental software updates to a client PC, thus eliminating a need for a user to manually do so, the user still faces the burden associated with the initial download and installation of the “push” application program.



US 6,317,761 B1

7

In addition, “push” application programs continue to increase in size, often considerably, as they provide added capabilities to a user. Downloading and then regularly updating a push application will reduce, sometimes considerably, the amount of disk space available to the user on his(her) client PC. Furthermore, “push” applications rely on periodically “pushing” large quantities of media content from a push server to the client PC and storing that content on the hard disk of that PC pending subsequent display. This content, depending on its volume, can consume inordinate amounts of hard disk space. Furthermore, advertisers have discovered, not surprisingly, that relatively few PC users will undertake any affirmative action, such as by downloading and installing an application program—almost regardless of its size, to receive advertisements and other such information.

Faced with these practical, and rather acute, deficiencies inhering in web advertising conventionally provided on either an interstitial or “push” basis, web advertisers have apparently relegated their efforts to displaying their advertisements on a banner-like approach, through real-time downloading and rendering of advertising HTML files. Here, the advertising files are sited on remote web servers, rather than being embedded within given web page HTML files, with appropriate HTML tags, which reference the ad files, being embedded into the web page files themselves. Such a tag specifies when and where, within the page, an advertisement is to appear.

To surmount the latency problems inherent in such banner-like advertisements, various proprietary media formats have appeared in the art. These formats employ increasingly sophisticated data compression, sometimes in conjunction with video and/or audio streaming. Rather than waiting for a media file to fully download prior to its being rendered, streaming permits content in a “streamed” media file to be presented in real-time to the user as that content arrives at his(her) client browser. While this approach clearly provides enhanced richness in content over that obtainable through a conventional banner and thus can heighten a “user experience”, it nevertheless relies, to its detriment, on a continuous real-time network connection existing to a remote web server.

Unfortunately, any network or server congestion which stops the download, even if temporary, can suspend, i.e., freeze, or totally halt the “streamed” media presentation to the user prior to its completion. This interruption, if noticeable and sufficiently long, will likely frustrate the user and degrade the “user experience”.

In spite of these drawbacks, particularly with respect to interstitial advertisements and push technology, and apparently for lack of a better alternative, most web advertising currently in use employs real-time streaming of graphic files with their content being rendered by the browser.

Web advertisements, like other forms of mass advertising, do generate revenue, often in the form of an on-going stream of payments to the host of the ads, in this case web site owners. Accurate user accounting is essential to ensure that an advertiser is not over- or under-charged given an extent to which an ad is actually disseminated. Hence, these payments are often tied to a function of the number of web users whom the ad reached. But with web advertisements, accurately ascertaining that number has been difficult and problematic at best, and, given a basic technique employed to do so, manifestly error-prone, thereby causing unreliable user counts and erroneous ad charges.

In particular and as conventionally employed, delivery of a web advertisement, such as, e.g., a streamed ad, is logged

8

as a “user impression” at a web server at an instant an advertising file(s), e.g., a streamed file, is served, rather than after the browser has completely rendered the advertisement to the user. Unfortunately, serving these ad files does not guarantee that these files will be ultimately and completely rendered by a client browser to a user. Consequently, web server generated “user impression” counts can be grossly over—or under-stated. For example, if a user navigates to a new content page after an advertisement has started playing but before that advertisement completes and, by doing so, prematurely terminated the advertisement, a full impression is nevertheless logged—erroneously—since that advertisement was completely served. Additional errors arise if a proxy server is situated between multiple client PCs situated on an intranet or a local area network (LAN) and a web advertisement server situated on the Internet (or other insecure public network). In this case, a request from one of the client PCs for the advertisement files will be routed to the proxy server, which, in turn, will direct that request onward to the advertisement web server. The latter, in response to the request, will serve one complete copy of the advertisement files to the proxy server. The resulting fetched advertisement files will be locally cached in the proxy server and, from there, provided to the requesting client PC. Should any of the other client PCs request the same files, the proxy server will provide these files, totally unbeknownst to the web server, from its local cache rather than directing a request from that other PC back to the web server. Hence, the web server will be totally oblivious to each additional instance in which the proxy server accessed the ad files from its local cache and disseminated the advertisement to any client PC other than that which first requested the ad. Inasmuch as some intranets situated behind a proxy server(s) can be rather extensive with tens or hundreds of thousands of individual client PCs, server-based user impression accounting based on copies delivered by a web server may, owing to the presence of proxy servers, be inordinately low and result in significant under-charges to the advertiser. As of yet, no solution apparently exists in the art that can provide accurate counts of “user impressions” of web advertisements.

Other conventional approaches aimed at reducing latency times associated with downloading content files through relatively slow speed communication links, e.g., modem connections, have involved development and use of new facilities within various programming languages. These approaches, most notably involving the Java and JavaScript programming languages, while helpful, still cause inefficient use of available link bandwidth and still constrain the size of the content files. These limitations arise from premature terminations of preloaded files whenever a user transitions to a new web page. Specifically, with these approaches, if a user activates a hotlink to transition to a new web page while an ad file is being downloaded but before the downloading has completed, then the downloading simply stops. The downloading will need to be re-started, but from the beginning of the file, the next time that particular ad file is requested. Hence, the time and bandwidth that has then been expended in downloading part of that ad file is completely wasted. In practice, many users tend to quickly navigate through a series of web pages until they reach a desired destination. Consequently, advertisers are constrained to again minimize content file sizes and hence “richness” of their advertisements in an effort to decrease a number of premature terminations per unit time and in doing so reduce latency caused by downloading duplicate sections of the same ad file. Therefore, these approaches have generally proven to be wholly unsatisfactory.

In view of the fundamental drawbacks associated with various web based advertising techniques known in the art, interstitial web advertising appears to hold the most promise of all these techniques. Yet, the limitations inherent in conventional implementations of interstitial advertising have effectively prevented this form of web advertising from effectively fulfilling its promise. Moreover, the deficiencies inherent in all known web advertising techniques have, to a significant extent, collectively inhibited the use of web advertising in general.

Thus, a pressing need exists in the art for a new web-based interstitial advertising technique which does not suffer from infirmities associated with such interstitial advertising techniques known in the art.

In that regard, this new technique should preferably not embed advertising HTML files within a web page. If this could be accomplished, then advantageously such a technique would likely provide considerable economies to advertisers in saved labor, time and cost in terms of both inserting advertisements into web page files, and later changing any of those advertisements. In addition, such a new technique should preferably function in a manner that is substantially, if not totally, transparent to a user and which neither inconveniences nor burdens that user. In particular, this new technique should preferably not require a user to download and install on his(her) PC a separate application program, let alone any update to it, specifically to receive web advertising, or perform any affirmative act, other than normal web browsing, to receive such advertising. Furthermore, this new technique should preferably be platform independent and, by doing so, operate with substantially any web browser on substantially any PC. Also, this new technique, when in use, should preferably not consume excessive hard disk space on a client PC. Moreover, to provide a pleasing "user experience", this new technique should render an ad fully and without any interruptions that might otherwise result from network and/or server congestion. Lastly, this new technique should provide proper accounting to an advertiser by accurately and validly ascertaining user impressions of fully rendered advertisements.

We believe that if such a new web-based interstitial advertising technique could be provided, then this technique, which should be both effective and desirable, may well achieve broad support and use by advertisers and acceptance by web users; hence, substantially expanding the use of web-based advertising in general.

SUMMARY OF THE INVENTION

Advantageously, our present inventive technique satisfies this need by overcoming the deficiencies associated with conventional web-based interstitial advertising techniques.

Our present invention accomplishes this, in accordance with our broad inventive teachings, by: completely "decoupling" advertising content from a web content page (also hereinafter referred to as a "referring" page); "politely" downloading advertising files, through a browser executing at a client computer, into browser caches (e.g., browser disk and RAM cache) at that computer and in a manner that is transparent to a user situated at the browser; and interstitially displaying advertisements through the browser in response to a user click-stream associated with normal user navigation across different web pages.

Specifically, our technique relies on embedding an HTML tag (which, where necessary, to distinguish this tag from other HTML tags, will also be referred to hereinafter as an "advertising tag") into a referring page. This tag contains

two components. One component effectively downloads, from a distribution HTTP (web) server and to an extent necessary, and then persistently instantiates an agent, implemented as a "light-weight" Java applet, at the client browser. This agent then "politely" and transparently downloads advertising files (media and, where necessary, player files), originating from an ad management system residing on a third-party advertising HTTP (web) server, for a given advertisement into browser disk cache (also in the case of media files into the browser RAM cache) and subsequently plays those media files through the browser on an interstitial basis and in response to a user click-stream. The other component is a reference, in terms of a web address, of the advertising management system from which the advertising files are to be downloaded. This latter reference totally "decouples" advertising content from a web page such that a web page, rather than embedding actual advertising content within the page itself—as conventionally occurs, merely includes an advertising tag that refers, via a URL, to a specific ad management system rather than to a particular advertisement or its content. The ad management system selects the given advertisement that is to be downloaded, rather than having that selection or its content being embedded in the web content page.

Advantageously, the agent operates independently, in the client browser, of the content in any referring web page. Once loaded and started, the agent executes in parallel, with standard browser functionality, continually and transparently requesting and downloading advertisements to browser cache residing in a client computer (e.g., personal computer—PC) and interstitially playing those advertisements.

In particular, once the agent is started, the agent politely and transparently downloads, through the client browser and to the browser cache, both media and player files, originating from the advertisement management server, for an advertisement that are needed to fully play content in that advertisement. The agent also monitors a click-stream generated by a user who then operates the browser. In response to a user-initiated action, e.g., a mouse click, which instructs the client browser to transition to a next successive content web page and which signifies a start of an interstitial interval, the agent, if all the media and player files are then resident on the client hard disk, plays the media files, through the browser and during that interstitial interval, directly from the browser cache. Advertisements are interstitially played typically in the order in which they were downloaded to the client browser. Interstitial play from browser cache advantageously permits previously cached content rich advertisements to be played through the browser without adversely affecting communication link bandwidth then available to the client browser. Thus, the full available link bandwidth can be used, while an advertisement is being played, to download a next successive content web page.

Employing a user click-stream to trigger play of cached advertisements frees the user, for receiving advertising, of any need either to undertake any affirmative action, other than normal web browsing, or to learn any new procedure; thus, advantageously imposing no added burden on the user.

Advantageously, the agent "politely" downloads advertisement media and player files, originating from the advertising server, to the browser cache, during what otherwise would be browser idle times, i.e., while a web page is being displayed to a user and the browser is waiting for user input. Caching advertisement files in this fashion advantageously circumvents variable latency and erratic (e.g., intermittent or

US 6,317,761 B1

11

suspended) play that frequently occurs with conventional streamed and static media delivered over the web.

At the start of an interstitial interval, the agent determines whether all the media and player files required to play a given advertisement (typically that having its so-called AdDescriptor file situated in a head of a play queue) then reside on the disk of the client PC or, with respect to media files, are resident in browser RAM cache. If so, the agent then accesses these files from the disk to “play” that advertisement. Since all the media and player files are then locally resident, the advertisement, from a user’s perspective, is immediately rendered from the client hard disk or browser RAM cache with essentially no downloading delay, thus providing a highly pleasing “user experience” with rich multi-media content approaching that obtainable through current CD-ROM based delivery. Thereafter, the agent returns control to the browser to permit the browser, if a next successive web page has been downloaded, assembled and ready to be rendered, to render that particular page to the user. If, however, an advertisement is prematurely terminated by a user, that advertisement (in terms of its AdDescriptor file) will remain in a play queue (with its media and player files remaining on the client hard disk or, in the case of media files, in browser RAM cache) and will be re-played from its beginning at the start of a next successive interstitial interval. Furthermore, if download of the media and player files for an advertisement were to be interrupted by a user click-stream, i.e., start of interstitial interval, the agent suspends further downloading until after the ensuing interstitial interval terminates. To conserve communication link bandwidth, the agent then resumes downloading of these files at a point it was suspended, rather than, as conventionally occurs, totally re-starting the download.

In accordance with our specific inventive teachings, the agent contains two applets: a Transition Sensor applet and an “AdController” applet. Only the Transition Sensor applet is itself associated with any content page. Though the AdController applet, once started, executes under the browser, it is not under the control of the browser itself.

The advertising tag is itself embedded in a content web page. The advertising tag, as one of its components, references a JavaScript file (which contains a “script”) stored on a distribution server. The JavaScript file, when executed, downloads and implements, through dynamic writing of applet tags, the Transition Sensor applet. This particular applet remains visually transparent to a user who displays, with his(her) browser, the HTML coding for that page. Specifically, when the JavaScript file is downloaded and the script it contains is then executed by the browser, the script dynamically writes a predefined number and combination of applet tags, i.e., which collectively form the Transition Sensor applet, into the retrieved web page content in lieu of the advertising tag. Subsequent execution of these tags, by the client browser, invokes the Transition Sensor applet. As discussed, the advertising tag also, as the other of its components, encapsulates a reference, i.e., a URL to a specific ad management server, typically sited on a third party advertising server, containing specific media, that collectively constitutes web advertisements, and accompanying player files.

In particular, when executed, the Transition Sensor applet instantiates an Applet Registry, which is used for inter-applet communication. Thereafter, the Transition Sensor applet determines whether the AdController applet has been downloaded to the browser disk cache or whether an updated version of this particular applet resides on a distribution server. If an updated version of this applet exists on the

12

distribution server relative to that previously downloaded to the browser disk cache or if this applet has not been download at all onto this cache, the Transition Sensor applet loads the AdController applet from the distribution server into the browser disk cache. The Transition Sensor applet then instantiates the AdController applet. Once this occurs, the Transition Sensor applet then establishes appropriate entries in the Applet Registry for itself and the AdController applet.

The Transition Sensor applet then passes the URL of the ad management system, as specified in the advertising tag, to the AdController applet in order for the latter applet to request delivery of an advertisement, specifically an associated AdDescriptor file, originating from that system. The system then selects the advertisement to be delivered and, via the third party advertising server, so informs the AdController applet by returning the requested AdDescriptor file. For a given advertisement, this particular file, which is textual in nature, contains a manifest, i.e., a list, of: file names and corresponding web addresses of all media files that constitute content for that advertisement and all player files necessary to play all the media files; an order in which the various media files are to be played; and various configuration and other parameters need to configure and operate the operation of each player in order for it to properly play a corresponding media file(s). The AdController then “politely” downloads, typically via the advertising distribution server, the associated media and player files, as specified in the AdDescriptor file—and to the extent they do not already reside on the hard disk of the client PC. As noted above, the Transition Sensor applet also monitors a click-stream produced by the current user to detect a user-initiated page transition and hence the start of an interstitial interval.

Advantageously, the AdDescriptor file implements a data abstraction that totally separates the media and player files from the referring web page thus assuring that the advertisement content itself remains completely independent of the content web page that invoked its presentation. This abstraction permits our technique to provide a highly effective, generalized and very flexible mechanism for delivering rich web advertisements, particularly those that require complex sets of media files and players. Through use of this abstraction, our technique is able to handle present and future media formats, regardless of their requirements, including proprietary streaming and other content delivery technologies that rely on Java applets as a delivery mechanism—all transparently to the user. Moreover, since the AdDescriptor file can specify media and player files for different browsers, operating systems and computing platforms then in use, our technique can readily function with a wide variety of different computing and browsing platforms.

The Transition Sensor and AdController applets are each implemented through appropriate Java classes and collectively persist, through storage in the browser disk cache, across different content pages within a site, across different web sites, and across successive browser sessions. Once either of these applets is completely downloaded, providing it is not subsequently flushed from the browser disk cache as the user navigates across web sites on the web, the files for that applet will be loaded from that cache, rather than being downloaded from the distribution server, the next time that applet is required, e.g., when the user next navigates, either during a current browser session or a subsequent session, to any content page that contains an advertising tag.

Whenever the client browser encounters a next successive page containing an advertising tag, then the browser will first and automatically inquire with the distribution server to



ensure that executable code for the Transition Sensor applet, if previously downloaded into the browser disk cache, has not been superseded by an updated version. If such an updated version then exists, the browser will collectively download updated files from the distribution server and replace, to the extent necessary, each Transition Sensor applet file residing in the browser disk cache with its updated version. Alternatively, if the Transition Sensor applet has not been previously downloaded into the browser disk cache, then the browser will download all the necessary files for the Transition Sensor applet from the distribution server into that cache. The Transition Sensor applet, once executing, will load, through the browser, the AdController applet. To do so, the browser will, if necessary, obtain an updated version, from the distribution server, in the same manner as it did for the Transition Sensor. As a result, any corrections or enhancements made to the agent (specifically the Transition Sensor and/or the AdController applets) since the agent was last downloaded to the client browser will be automatically and transparently, from a user perspective, distributed to that browser and downloaded into the browser disk cache the next time the browser encounters a web page containing an advertising tag. By operating in this fashion, the user is totally and advantageously relieved of any need to: both initially load and install an application program to obtain advertising and/or later update that program.

Furthermore, the agent advantageously persists and functions transparently in background, independent and transparent to user navigation across pages on a common web site and across web sites. The agent effectively implements a background process which runs in parallel with and is transparent to normal HTML and HTTP operations implemented by the client browser.

Moreover, in sharp contrast to conventional server-based accounting of web advertisements, our inventive technique provides highly accurate client-side accounting of each user impression. Each log entry, produced by the AdController applet, specifies a successful presentation of a complete advertisement at a client browser. This entry may include a source of the ad content, i.e., in terms of the URL of the associated ad management system, a title of the advertisement and the URL of the referring web page. Other client-side information can be measured and included in each entry, such as: an amount of time during which the advertisement was rendered by the browser (presumably during which the user dwelled on the advertisement); as well as an identification, in terms of a URL, of a content web page to which the user next navigated (particularly if the user reached that page through a hotlink displayed in the advertisement). Subsequently, the AdController applet uploads the log entries to the advertising server. These entries will be collectively processed, as needed, to permit shared ad revenues from web-based advertisers to be properly allocated among different web page content providers.

Advantageously, our inventive technique, by totally decoupling referring web page content from its corresponding advertising content, easily permits an advertiser to change or update any of its advertisements by just modifying, as needed, appropriate media and AdDescriptor files that reside in the third-party advertising management system. Since a referring web page merely incorporates an advertising tag totally devoid of advertising content, no changes whatsoever need to be made to that page. Hence, use of our inventive technique substantially reduces the burden, time and cost associated with maintaining and updating web-based advertising over that conventionally required.

As a feature, our inventive technique advantageously implements, in conjunction with its persistent agent approach, multi-threaded pipelining. By processing each different advertisement as a different thread, each one of a sequence of different processing operations can be performed, effectively on a pipe-lined parallel basis, on different sequentially occurring advertisements, thereby enhancing a rate (increasing throughput) at which advertisements can be queued for playback. In addition, through such pipe-lining, logging of a fully presented advertisement can occur as a last operation in a pipeline and essentially in parallel either with: presentation of cached advertisement having its AdDescriptor file situated in the play queue immediately behind that for the just presented advertisement, or downloading and caching of a next successive advertisement.

BRIEF DESCRIPTION OF THE DRAWINGS

The teachings of the present invention can be readily understood by considering the following detailed description in conjunction with the accompanying drawings, in which:

FIG. 1A depicts the correct alignment of the drawing sheets for FIGS. 1B and 1C;

FIGS. 1B and 1C collectively depict a high-level block diagram of an illustrative client-server distributed processing environment, implemented through the Internet, which embodies the teachings of our present invention, along with, as pertinent to the invention, basic inter-computer actions that occur in that environment and associated client processing operations;

FIG. 1D depicts the correct alignment of the drawing sheets for FIGS. 1E and 1F;

FIGS. 1E and 1F collectively depict the same environment as shown in FIGS. 1B and 1C but showing a detailed version of agent download/instantiate/execute operations shown in the latter figures;

FIG. 2 depicts the correct alignment of the drawing sheets for FIGS. 2A and 2B;

FIGS. 2A and 2B collectively depict generalized web page HTML code, specifically inclusion of advertising tag, which transparently invokes our invention, and changes which our invention dynamically makes to that code, specifically substitution of Transition Sensor applet for tag to yield page, in order to download and render web advertisements;

FIG. 3 depicts a high-level block diagram of client PC shown in FIGS. 1B and 1C, and 1E and 1F;

FIG. 4 depicts a simplified high-level block diagram of application programs resident within client PC shown in FIG. 3;

FIG. 5 depicts a high-level block diagram of AdController agent shown in FIG. 4, which implements our present invention;

FIG. 6 depicts the correct alignment of the drawing sheets for FIGS. 6A and 6B;

FIGS. 6A and 6B collectively depict a high-level flowchart of processing operations performed by AdController agent shown in FIG. 5;

FIG. 7 depicts a high-level block diagram of basic processing threads that implement AdController applet which, as shown in FIG. 4, forms part of AdController agent;

FIG. 8 depicts a high-level flowchart of processing operations performed by AdController applet shown in FIG. 7;



FIG. 9 depicts the correct alignment of the drawing sheets for FIGS. 9A and 9B;

FIGS. 9A and 9B collectively depict a flowchart of processing operations 900 performed by AdController applet 424, shown in FIG. 7, specifically for processing an advertisement;

FIG. 10 depicts inter-applet events that occur within AdController agent 420 during execution of Transition Sensor applet 422;

FIG. 11 depicts a high-level block diagram of basic processing threads that implement Transition Sensor applet 422 which, as shown in FIG. 4, forms part of AdController agent 420;

FIG. 12 depicts a high-level flowchart of processing operations 1200 performed by Transition Sensor applet 422 shown in FIG. 11;

FIG. 13 depicts a high-level block diagram of Ad Loader process 1300 which can be used to provide advertiser control over various functions, for advertisement play and logging, implemented by AdController applet 424;

FIG. 14 depicts a high-level block diagram of Ad Pipeline 545 that is implemented by and forms part of AdController applet 424 shown in FIG. 4;

FIG. 15 depicts a high-level block diagram of Ad Producer process 1500 that is executed by Ad Pipeline 545 shown in FIG. 14;

FIG. 16 depicts a high-level block diagram of Ad Location process 1600 that is also executed by Ad Pipeline 545 shown in FIG. 14;

FIG. 17 depicts a high-level block diagram of Ad Downloader process 1700 that is also executed by Ad Pipeline 545 shown in FIG. 14;

FIG. 18 depicts a flowchart of stop method 1800 invoked by Transition Sensor applet 422 shown in FIG. 4;

FIG. 19 depicts a flowchart of start method 1900 invoked by Transition Sensor applet 422 shown in FIG. 4; and

FIG. 20 depicts contents of actual illustrative AdDescriptor file 2000 for use in interstitially rendering a PointCast type Java advertisement through our present invention.

To facilitate understanding, identical reference numerals have been used, where possible, to designate identical elements that are common to the figures.

DETAILED DESCRIPTION

After considering the following description, those skilled in the art will clearly realize that the teachings of our present invention can be utilized in any networked client-server environment in which advertising or other information is to be presented to a user during interstitial intervals, i.e., during a transition between successively displayed web pages. Such an environment can encompass the Internet or an intranet, or any client-server environment in which a client browser (regardless of whether that browser executes on a dedicated client computer or not) is used to access and download web pages or, more generally speaking, files through a network communication channel (link) from a server (again regardless of whether that server executes on a dedicated computer or not). In that regard, the server can be a separate software application which executes on any computer in the networked environment, even if that computer is itself a client to another server in the network.

For purposes of simplicity and to facilitate reader understanding, we will discuss our present invention in the illustrative context of use in rendering interstitial web-based

advertisements to a client personal computer (PC) connected to the Internet, where specifically a client browser executing in the PC is used to download and render web pages from a remote networked Internet accessible web server. Clearly, after considering the ensuing description, anyone skilled in the art will readily appreciate how the teachings of our invention can be easily incorporated into any client-server or other similar distributed processing environment in which a client can encompass not only a specific computer connected to a network but a software process that possesses network connectivity to another such process and requests information from and, in response, obtains information supplied by the latter.

We will first present an overview of our invention, particularly in the context of its use with an Internet web browser in a client PC, followed by describing each basic component of its implementation.

A. Overview

A general deployment of our invention in an Internet environment is collectively shown in FIGS. 1B and 1C, with a detailed view of a portion of the inter-processor agent download/instantiation operations 50 shown in these figures being depicted in FIGS. 1E and 1F. The correct alignment of the drawing sheets for FIGS. 1B and 1C, and 1E and 1F is shown in FIGS. 1A and 1D, respectively. FIGS. 2A and 2B, for which the correct alignment of the drawing sheets for these figures is shown in FIG. 2, collectively depicts generalized web page HTML code which transparently invokes our invention, and changes which our invention dynamically makes to that code in order to download and render web advertisements. For a understanding, the reader should simultaneously refer to FIGS. 1B and 1C, 1E and 1F, and 2A and 2B throughout the following discussion.

As shown, client PC 5, upon which client browser 7 executes, is connected through communication link 9 to Internet 10. Browser 7 is a conventional web browser, such as Internet Explorer or Netscape Navigator commercially available from Microsoft Corporation or Netscape Corporation, respectively. Preferably, for reasons that will shortly become clear, that browser should preferably support dynamic writing of applet tags. Though, for ease of illustrating inter-computer actions, we depicted Internet 10 as having portions 10<sub>A</sub> and 10<sub>B</sub>, we will collectively refer to both portions as simply Internet 10. Web server 13, connected, via link 11, to Internet 10 represents any web HTTP (hypertext transfer protocol) server. This server, in response to a request from web browser 7 to fetch a specific file, downloads that file, using conventional TCP/IP protocols (transmission control protocols/internet protocols), through the Internet to browser 7. Browser 7 will, in turn, render that file typically on a monitor to a user situated at the client PC.

Advertising distribution HTTP server (also referred to as "agent" server) 15 is connected, via communications link 17, to Internet 10 and stores files that collectively implement a predefined agent, specifically, a light weight Java applet. This agent (referred to herein as the "AdController" agent) transparently pre-loads itself, as well as media rich advertising content, into a local hard disk cache associated with the browser ("browser disk cache") on client PC 5. Server 15 downloads the AdController agent in a manner to be described below, to client browser 7. This agent, once instantiated and started, then transparently and politely downloads (actually pre-loads) advertisements into the browser disk cache, and subsequently plays each of those advertisements, on an interstitial basis, in response to a click stream generated by the user as (s)he navigates, through use

of browser 7, between successive web pages. Such hard disk caching advantageously circumvents variable latency and erratic play associated with conventional streamed and static media delivered over the Internet. The agent enables rich advertising to be presented in a highly-controlled fashion, resulting in user experiences approaching that of CD-ROM.

Third-party ad HTTP server 20, connected to Internet 10 via, e.g., communications links 18 and 23, hosts ad management system 25. In essence and as discussed in detail below, this system, in response to a request originating from the AdController agent executing in browser 7, selects a given advertisement and then downloads, in a “polite” manner controlled by the agent, media and player files that form that advertisement to the agent for storage in the browser disk cache. Inasmuch as Java applets are currently restricted under constraints inherent in the Java programming language itself to retrieving files from an identical Internet host that served the applet itself, the request for an advertisement to system 25 as well as resulting media and player files served by system 25 are routed through agent server 15 as a proxy server.

Advantageously, our inventive technique completely “decouples” advertising content from a web content page (also hereinafter referred to as a “referring” page). This, in turn, permits our technique to render media-rich advertisements without requiring inclusion of any advertising content into a referring web page. This “decoupling” is effectuated through inclusion of an HTML tag into a content web page, which when the latter is downloaded, interpreted and executed by the browser, effectively loads and instantiates the agent and then retrieves advertisement files from an ad management system specified in the tag. Thus, advertising files (both media and player files) can be maintained totally independently of their referring web page(s), with advantageously any changes made to the former having no effect on HTML coding contained in the latter.

In particular, HTML tag 40 (which, where necessary, to distinguish this tag from other HTML tags, will also be referred to hereinafter as an “advertising tag”) is embedded by a content provider(s) into HTML code that constitutes each referring web page, e.g., here page 35. Generally, the position of this tag relative to existing HTML code (shown as HTML code portions 35<sub>A</sub> and 35<sub>B</sub> in FIGS. 2A and 2B) for this page is not critical. Advantageously, very rarely, if ever at all, do any changes need to be made to these code portions to accommodate the tag. As shown and as reproduced in Table 1 below, this tag, which typically consumes one line in a web page, implements a script.

```
<SCRIPT SRC=http://unicast.com/loadad.js>
AdServer="http://AdManagement system"
</SCRIPT>
```

TABLE 1—ADVERTISING TAG

One portion of the advertising tag (“SRC=http://unicast.com/loadad.js”), when executed by the browser, downloads a JavaScript file (named “loadad.js”) from the agent server. This file, in turn, is then interpreted and executed, as a script, by the browser. The effect of executing this script, as symbolized by block 200 shown in FIGS. 2A and 2B, is to substitute applet tags, dynamically written by the script, into the referring web page in lieu of advertising tag 40 so as to form a modified web page, here referring content page 35', residing in the browser disk cache. The script, by invoking a feature associated with dynamic writing, completely hides these tags from view should the user then display HTML source code for page 35' with his

browser. This, in turn, hinders the user, to a certain degree, from readily ascertaining the source of the agent and ad management systems. Collectively, these applet tags form Transition Sensor applet 210. This script, as described in detail below and is reproduced in Table 2 below, when interpreted and executed by a Java virtual machine (Java interpreter) resident in the browser persistently loads and then instantiates the Transition Sensor itself which, in turn, loads and instantiates the remainder of the agent in the client browser.

```
<applet                                code=
    "com.unicast.adcontroller.tools.TransitionSensor"
codebase="http://www.unicast.com/java/classes/"
align="baseline" width="0" height="0" name=
    "TransitionSensor"
archive="adcontroller.jar">
<param name="adURL"
value="http://www.unicast.com/media/fireworks_01_
ad_descriptor.txt">
<param name="cabbase" value="adcontroller.cab">
</applet>
```

TABLE 2—TRANSITION SENSOR APPLET

The value of attribute CODE in the Transition Sensor applet specifies a Java executable that will be executed by the client browser, when it renders this applet, to launch the Transition Sensor. The executable, implemented through an appropriate Java class, was originally compiled from its associated Java source code file. Tags labeled “<WIDTH>” and “<HEIGHT>” jointly specify a rectangular portion of a web page, as displayed by browser 7, in which the applet will be rendered. Since, here that portion is non-existent, nothing will be rendered. Applets, such as this one, can be delivered transparently over the Internet to the client PC and require no user-assisted installation.

Another portion of the advertising tag (“AdServer=“http://AdManagement system”) references a URL of a particular ad management system (where “AdManagement\_system” represents a web address (URL) of that particular system), here illustratively system 25, from which the agent is to download an advertisement. As will be seen below, the Transition Sensor applet, during its execution, passes this URL, as part of an advertising download request, to the remainder of the AdController agent to subsequently download appropriate advertising files, also as described below, from that system necessary to interstitially play an advertisement.

If advertisements are to play on client browsers (specifically Microsoft Internet Explorer version 3) that do not support dynamic writing of applet tags, then applet 210 would need to be inserted by content providers into each referring web page in lieu of advertising tag 40. Unfortunately, Transition Sensor applet 210 identifies both the agent server, and an actual advertisement in terms of a URL of its source components (through contents of an “AdDescriptor” file—which will be discussed in detail below—specified in this applet). Since browser technology continues to rapidly advance with most users continually upgrading their browsers, most browsers now in use, and in a very short time nearly all such browsers, will support such dynamic writing. Hence, we see little, and very shortly essentially no need, to embed applet 210 into any referring web pages, thus minimizing ad insertion cost, effort and time while restricting disclosure of the agent server and advertisement source information.

The agent, during its execution, “politely” and transparently downloads advertising files (media, and where necessary player files), originating from ad management system 25 for a given advertisement into browser disk cache (with the media files also being written into browser RAM cache) and subsequently plays those media files through the browser on an interstitial basis and in response to the user click-stream.

Advantageously, the agent operates independently, in the client browser, of the content in any referring web page. Once loaded and started, the agent executes in parallel, with standard browser functionality, continually and transparently requesting and downloading advertisements to a browser disk cache residing on a local hard disk (“browser disk cache”), as well as in the case of media files into browser RAM cache, in a client computer (e.g., personal computer—PC) and interstitially playing those advertisements.

Now, with the above in mind and specific reference to FIGS. 1B and 1C, we will now describe the basic inter-computer actions associated with use of our invention, as well as the basic attendant processing steps that occur in the client PC.

To begin a browsing session, the user first invokes client browser 7. Once the browser is executing, the browser obtains, as an initial web page—selection of this page being referenced by numeral 31, an address either of a prior so-called “default” content page previously specified by the user and having its URL stored in the browser or of a content page manually entered by the user. The client browser then issues, as symbolized by block 33, a request to fetch a file for that page; with the request containing a URL of that page (i.e., its complete web address including its file name). We assume for simplicity that the file for that page resides on web server 13. We also assume that page 35 is being requested which will invoke an associated interstitial advertisement in accordance with our invention. In response to the request routed to server 13—as symbolized by line 34, this particular server downloads, as symbolized by line 36, to client PC 5 a file for page 35, where the coding stored in this file contains advertisement tag 40. Illustrative contents of this tag are shown in dashed block 45, as well as in FIGS. 2A and 2B.

Once this file is received as shown in FIGS. 1B and 1C, browser 7 interprets and then executes, as symbolized by block 52, the HTML code in page 35, which includes tag 40 and thus undertakes the actions shown in agent download/instantiate/execute operations 50. These operations eventually result in the AdController agent being downloaded, instantiated and started in the client browser. Generally speaking, the browser in response to executing the advertising tag, issues a request, as symbolized by line 54, to agent server 15 to download the AdController agent. Through various inter-process operations, as shown in further detail in FIGS. 1E and 1F and which will be described below shortly, server 15 accesses and downloads, as symbolized by line 56, the needed files to install the AdController agent to execute under browser 7 on the client PC. Once files for the agent are downloaded to the browser disk cache on the client PC, the browser then instantiates and starts the agent executing, as symbolized by block 58. Operations 50 effectively conclude once the agent begins executing.

Now referring to operations 50 as shown in further detail in FIGs. 1E and 1F, upon entry into these operations, browser 7 executes, as symbolized by block 110, advertising tag 40. The browser then issues a request, as symbolized by

line 115, to agent server 15, to download a JavaScript file (named, e.g., “loadad.js”) specified in the request. This file is specified as the first portion of the advertising tag. In response to this request, server 15 downloads, as symbolized by line 120, this particular file onto browser 7 where that file is cached appropriately. Once the file is fully downloaded, it is interpreted and executed by a Java virtual machine (a Java interpreter integrated into the browser and which generates code compatible with and executable by the browser). As indicated by block 125, the browser then executes the interpreted code for the script which, in turn, dynamically writes applet tags 13 in the manner generally shown in FIGS. 2A and 2B and described above—into web page 35 in lieu of the advertising tag. These tags, which collectively form Transition Sensor applet 210, include a reference to a specific ad management system as specified in the second portion of advertising tag 40.

Once these tags are dynamically written into content web page 35 (to yield modified version 35' shown in FIGS. 2A and 2B), Transition Sensor applet 210 is instantiated and then executed. In particular, browser 7 determines whether executable code for the Transition Sensor applet has been previously downloaded to the browser disk cache. If this code has not been downloaded or an updated version of this code exists on agent server 15, the browser issues, as symbolized by line 130, a request to download a latest version of the Transition Sensor executable code from the agent server. Server 15, in response to this request, downloads, as symbolized by line 135, file(s) for the latest version of the transition sensor code to the browser which, in turn, stores these file(s) into the browser disk cache. Thereafter as symbolized by block 140, the browser instantiates and starts execution of the Transition Sensor applet. This latter applet, as part of its initial execution, instantiates an Applet Registry. This registry provides a mechanism, within the agent, for inter-applet communication between the constituent Transition Sensor and AdController applets.

Thereafter, the Transition Sensor applet attempts to load, also as symbolized by block 140, the AdController applet, through the browser, from the browser disk cache. To do so, the browser first determines whether the AdController applet has been downloaded to the browser disk cache or whether an updated version of this particular applet resides on agent server 15. If an updated version of this applet exists on the agent server relative to that previously downloaded to the browser disk cache or if the AdController applet has not been download at all into this cache, the browser issues a request, as symbolized by line 150, to download a latest version of the AdController applet from agent server 15. Server 15, in response to this request, downloads, as symbolized by line 155, file(s) for the latest version of the AdController applet to the client browser which, in turn, stores these file(s) into the browser disk cache. Lastly, as symbolized by block 160, the Transition Sensor applet then instantiates and starts the AdController applet; and thereafter establishes appropriate entries in the Applet Registry for itself and the AdController applet.

Returning to FIGS. 1B and 1C, once operations 50 have completed, such that the agent is executing under browser 7, the AdController applet issues, as symbolized by block 60, a request, via agent server 15, to download an AdDescriptor file from the ad management system, e.g., ad management system 25, specified in advertising tag 40. This request contains the URL of the ad management system contained in advertising tag 40. Currently, Java applets are restricted under constraints inherent in the Java programming language itself to retrieving files from an identical Internet host



US 6,317,761 B1

21

that served the applet itself. As such, rather than directing this request to advertising server **20**, on which ad management system **25** resides, this request, as symbolized by line **62**, is addressed to agent server **15**, which serves as a proxy server between client PC **5** and advertising server **20**. Both the request and resulting advertising (including media and player) files will be served to the client PC through agent server **15**. As such, once the request has been received by the agent server, this server passes the request onward, as symbolized by line **64**, to advertising server **20**.

In response to this request for an AdDescriptor file, ad management system **25** then selects a specific advertisement to be delivered to client PC **5**. This selection can be selected on a predefined or random basis, or based on user preference or other user-specific information previously collected from and associated with the user then operating browser **7**. Such user-specific information, such as prior buying patterns, could have been appropriately pre-collected at the client PC, previously uploaded to ad management system **25** and processed there such that, upon receipt of the AdDescriptor request, system **25** would then select and download an appropriate advertisement specifically targeted to the user then situated at the client PC. In any event, once system **25** selects the advertisement, through whatever selection metric it employs, the corresponding AdDescriptor file is then downloaded, as symbolized by line **66**, to agent server **15** (here being a proxy server) which, in turn, as symbolized by line **68**, supplies that file to the AdController agent then executing under web browser **7**.

To digress slightly, for the selected advertisement, the AdDescriptor file is a text file that contains a manifest, i.e., a list, of file names and corresponding network locations (URLs) at which these files reside, and player instructions and configuration parameter values necessary to play the entire advertisement through web browser **7** to the user. FIG. **20** shows contents of typical AdDescriptor file **2000** for a PointCast Java advertisement. Specifically and as shown in section **4C** of file **2000**, this AdDescriptor file lists file names with partial addresses on the ad management system of all media files that constitute content for that advertisement, and, in section **1** of this file, all Java player files necessary to play all the media files. This file also respectively specifies, here shown in sections **3** and **4B**, an order in which the various media files are to be played, and various configuration parameters needed to properly configure the operation of each player to play each corresponding media file.

The AdDescriptor file implements a data abstraction that totally separates the media and player files from the referring web page, here page **35**, thus assuring that the advertisement content itself remains completely independent of the content web page that invoked its presentation. This abstraction permits our technique to provide a highly effective, generalized and very flexible mechanism for delivering rich web advertisements, particularly those that require complex sets of media files and players. Through use of this abstraction, our inventive technique can handle present and future media formats, regardless of their requirements, including proprietary streaming and other content delivery technologies that rely on Java applets as a delivery mechanism—all transparently to the user. Moreover, the AdDescriptor file can contain separate listings (though not contained in file **2000** shown in FIG. **20**) that delineate media and player files for different browsers, client operating systems or computing platforms (to the extent any of these require different versions of the media and/or player files) then in use. As such, our technique can readily function with a wide variety of different client computers and browsing platforms.

22

Once the AdDescriptor file is downloaded to the client PC, via agent server **15**, the AdController then “politely” downloads, as symbolized by block **70** shown in FIGS. **1B** and **1C**, into the browser disk cache each media and player file, as specified in the AdDescriptor file—to the extent that file does not already reside on the hard disk of the client PC. Through so-called “polite” downloading, media and player files are downloaded to browser **7** during browser idle time intervals, with the downloading suspended during each ensuing interstitial interval after the user instructs browser **7** to navigate to a new content web page. In this manner, while a fully downloaded advertisement is interstitially played from browser cache, the new content page is downloaded over the full bandwidth of communications link **9**. Advantageously, the communications link is freed during each interstitial interval to just carry web page content, thereby expediting download of content pages. If, due to the occurrence of an interstitial interval, the AdController applet suspends downloading of an advertisement file, then upon termination of this interval, this applet then resumes downloading at a location in that file at which downloading had stopped, thus conserving communication bandwidth and reducing download time.

In particular, as part of the operations symbolized by block **70**, the AdController applet determines which files, of those listed on the AdDescriptor, do not then reside on the hard disk of client PC **5**. Once it has made that determination, this applet issues a request, as symbolized by line **72**, to agent server **15**, to fetch a first one of these files. The agent server, again serving as a proxy server, issues a request, as symbolized by line **74**, to fetch this file from a networked server, anywhere on Internet **10**, on which that file resides. For simplicity, we assume that all such files reside on server **20** and are accessible through ad management system **25**. Hence, system **25**, via server **20**, issues a response, as symbolized by line **76** to agent server **15**, containing this first advertisement file. The agent server, in turn and as symbolized by line **78**, downloads this particular file to client browser **7** for storage in the browser disk cache. Downloading of advertisement files continues in this manner until, as symbolized by line **88**, a last required file for the advertisement has been downloaded, via agent server **15**, to the browser disk cache on client PC **5**.

As the advertisement files for a common advertisement are being downloaded, the Transition Sensor applet also monitors, as symbolized in block **90**, a click-stream produced by the current user so as to detect a user-initiated page transition. Once such a transition occurs, usually caused by a user engendered mouse click, and hence an interstitial interval starts, the AdController applet plays, also as symbolized by block **90**, a fully cached advertisement (assuming all its media and player files have been downloaded) in the manner specified in its associated AdDescriptor file and using the players specified therein. Also, at the inception of the interstitial interval, the browser issues, also as symbolized by block **90**, a request to fetch the next successive web page to which the user desires to transition. Once the advertisement has fully played, or until the next successive content web page is fully downloaded and assembled, or a user has closed an advertisement window, whichever occurs first (assuming the AdDescriptor file specifies that the advertisement can be prematurely terminated), then control is returned, as symbolized by path **94**, to the client browser to await completion of the download and interpretation of HTML code that forms that next content page and subsequent execution, of an advertising tag therein to invoke agent download/instantiate/execute operations **50** for that page; and so forth.

The Transition Sensor and AdController applets are each implemented through appropriate Java classes and collectively persist, through storage in the browser disk cache, across different content pages within a site, different web sites, and successive browser sessions. Once either of these applets is completely downloaded through operations 50, providing that applet is not subsequently flushed from the browser disk cache as the user navigates across web sites on the web, the files for that applet will be loaded from that cache, rather than being downloaded from agent server 15, the next time that applet is required, e.g., when the user next navigates, either during a current browser session or a subsequent session, to any successive content page that contains advertising tag 40.

Whenever client browser 7 encounters a next successive content page containing advertising tag 40, then the browser, will first and automatically inquire with agent server 15 to ensure that executable code for the Transition Sensor applet, if previously downloaded into the browser disk cache, has not been superseded by an updated version. If such an updated version then exists, the browser will collectively download updated files from the agent server and replace, to the extent necessary, each Transition Sensor applet file residing in the browser disk cache with its updated version. Alternatively, if the Transition Sensor applet has not been previously downloaded into the browser disk cache, then the browser will download all the necessary files for the Transition Sensor applet from the agent server into that cache. The Transition Sensor applet, once executing, will load, through the browser, the AdController applet. To do so, the browser will, if necessary, obtain an updated version, from the agent server, in the same manner as it did for the Transition Sensor. As a result, any corrections or enhancements made to the agent (specifically the Transition Sensor and/or the AdController applets) since the agent was last downloaded to the client browser will be automatically and transparently, from a user perspective, distributed to that browser and downloaded into that disk cache the next time the browser encounters a web page containing an advertising tag. By operating in this fashion, the user is totally and advantageously relieved of any need to: both initially load and install an application program to obtain advertising and/or later update that program.

Specifically, cross page persistency of the Transition Sensor agent is accomplished by using a Java "singleton" design. A singleton design allows only a single object to ever be created and is accomplished by declaring a Java class as static. Since all applets run in a same instance of a Java virtual machine, therefore all applets and their associated code share all static class variables. A static Applet Registry class is instantiated automatically by the Transition Sensor applet at its run time and, by implementing the Applet Registry, provides all inter-applet communication between the Transition Sensor and the AdController applets and their threads. The Applet Registry class implements a "loadAdController" method which, in turn, instantiates the persistent AdController applet. Through this method, the Transition Sensor applet downloads the AdController applet only if the latter applet has either been updated, relative to that version of this applet then resident in the browser disk cache, or does not then reside on the browser disk cache. The AdController applet then instantiates all its own threads that collectively implement transparent advertisement downloading and play mechanisms.

The AdController applet is itself created by an Applet Registry singleton object and creates all other objects that collectively constitute a run time agent execution module.

This applet extends standard applet class definitions by over-riding standard Java applet init (initialize), start, run, stop and destroy life cycle methods, conventionally implemented in the client browser, with corresponding substitute methods. The substitute stop method ensures that a traditional response provided by the browser of halting execution for either the AdController applet does not occur whenever the browser calls the stop method to terminate the lifecycle of this applet; hence, advantageously providing persistence to the agent across successive content pages. Consequently, the agent continues executing until the user terminates execution of (closes) the browser itself.

Thus, the agent persists and functions transparently in background, independent and transparent to user navigation across pages on a common web site and across web sites. In that regard, the agent effectively implements a background process which runs in parallel with and is transparent to normal HTML and HTTP operations implemented by the client browser.

To significantly simplify the description and the accompanying drawings, we have intentionally omitted from this discussion specific Java classes that constitute the AdController agent as well as, to increase a rate at which advertisements can be queued for playback, an accompanying software architecture for processing these classes on a multi-threaded pipelined basis. Such details are conventional in nature; hence, their use in implementing our present invention would be readily apparent to any one skilled in the art.

B. Client PC

FIG. 3 depicts a block diagram of client PC 5.

As shown, the client PC comprises input interfaces (I/F) 320, processor 340, communications interface 350, memory 330 and output interfaces 360, all conventionally interconnected by bus 370. Memory 330, which generally includes different modalities, including illustratively random access memory (RAM) 332 for temporary data and instruction store, diskette drive(s) 334 for exchanging information, as per user command, with floppy diskettes, and non-volatile mass store 335 that is implemented through a hard disk, typically magnetic in nature. Mass store 335 may also contain a CD-ROM or other optical media reader (not specifically shown) (or writer) to read information from (and write information onto) suitable optical storage media. The mass store stores operating system (O/S) 337 and application programs 400; the latter illustratively containing browser 7 (see, e.g., FIGS. 1B and 1C) which implements our inventive technique. O/S 337, shown in FIG. 3, may be implemented by any conventional operating system, such as the WINDOWS NT, WINDOWS 95, or WINDOWS 98 operating system ("WINDOWS NT", "WINDOWS 95" and "WINDOWS 98" are trademarks of Microsoft Corporation of Redmond, Wash.). Given that, we will not discuss any components of O/S 337 as they are all irrelevant. Suffice it to say, that the browser, being one of application programs 400, executes under control of the O/S.

Incoming information can arise from two illustrative external sources: network supplied information, e.g., from the Internet and/or other networked facility, through communication link 9 to communications interface 350, or from a dedicated input source, via path(es) 310, to input interfaces 320. Dedicated input can originate from a wide variety of sources, e.g., an external database. In addition, input information, in the form of files or specific content therein, can also be provided by inserting a diskette containing the information into diskette drive 334 from which client PC 5, under user instruction, will access and read that information

from the diskette. Input interfaces **320** contain appropriate circuitry to provide necessary and corresponding electrical connections required to physically connect and interface each differing dedicated source of input information to client PC **5**. Under control of the operating system, application programs **400** exchange commands and data with the external sources, via network connection **9** or path(es) **310**, to transmit and receive information typically requested by a user during program execution.

Input interfaces **320** also electrically connect and interface user input device **395**, such as a keyboard and mouse, to client PC **5**. Display **380**, such as a conventional color monitor, and printer **385**, such as a conventional laser printer, are connected, via leads **363** and **367**, respectively, to output interfaces **360**. The output interfaces provide requisite circuitry to electrically connect and interface the display and printer to the computer system.

Furthermore, since the specific hardware components of client PC **5** as well as all aspects of the software stored within memory **335**, apart from the modules that implement the present invention, are conventional and well-known, they will not be discussed in any further detail. Generally speaking, agent server **15** and third-party ad server **20** each has an architecture that is quite similar to that of client PC **5**.

C. Software

1. Application programs **400**

FIG. **4** depicts a simplified high-level block diagram of application programs **400** resident within the client PC.

As shown, the application programs, to the extent relevant, contain browser **7** and resident JAVA player files **410**, i.e., files for JAVA media players that have previously been installed onto the hard disk of the client PC. These players may illustratively include audio, streaming audio, video and multi-media players.

Browser **7** contains AdController agent **420**, when it has been fully loaded for execution into browser cache, browser disk cache **430** and Java virtual machine **440** (which has been discussed above to the extent relevant). As noted, this agent persists whenever the user causes browser **7** to transition across different web content pages or different web sites, and functions independently and transparently of any such pages and sites. The AdController agent includes applet registry **426** for facilitating inter-applet communication within the agent.

The AdController agent contains two applets: Transition Sensor applet **422** and AdController applet **424** (also referred to as applet **210** in FIG. **2B**). As discussed above, the Transition Sensor applet accomplishes three basic functions. First, this applet loads, instantiates and starts the AdController applet. Second, the Transition Sensor applet communicates an Internet address of an advertising server, here server **20**, to request an advertisement, specifically an AdDescriptor file therefor, that is to be downloaded and subsequently presented. Lastly, the Transition Sensor applet, through associated click-stream monitoring (performed by a Transition Sensor implemented by this applet), determines when a user situated at client browser **7** undertakes an affirmative action, such as, e.g., causing a mouse click, to request a next successive web page be downloaded and rendered, and so notifies the AdController agent of that event. This event signals a start of an ensuing interstitial interval.

AdController applet **424**, which is not embedded in any content page, executes under but is not controlled by browser **7**. This applet, also as discussed above, accomplishes several basic functions. First, it creates S all other

objects that collectively form a run time agent execution module for the agent. As noted above, this includes extending standard Java applet class definitions by over-riding standard Java applet init, start, run, stop and destroy life cycle methods. Second, the AdController applet “politely” downloads advertising (including media and, where necessary, player) files, through the client browser executing at a client computer, into browser disk cache and in a manner that is transparent to a user situated at the browser. Lastly, the AdController applet interstitially plays advertisements through the client browser in response to the user click-stream associated with normal user navigation across different web pages.

Browser disk cache **430** stores downloaded AdDescriptor files **433** and accompanying and downloaded media and player files **437**.

2. AdController agent **420**

FIG. **5** depicts a high-level block diagram of AdController agent **420**.

As shown, the agent specifically contains Transition Sensor applet **422**, AdController applet **424** and applet registry **426**.

As discussed generally above, the Transition Sensor applet implements, as one of its functions, a transition sensor which detects, through user navigation click-stream monitoring, a user-initiated transition to a new web page, and produces, in response, a corresponding Transition Sensor event. Such a transition occurs in response to an actual user initiated mouse click or key depression to activate a hotlink appearing on a currently displayed content page in order to move to a new content page, either on the same site or on another site. Another such transition occurs whenever a stored history of web pages just visited by the user changes state. The latter is sensed by a JavaScript function that monitors a history stored in browser disk cache **430** of visited web page URLs and generates an event whenever the history changes state. For ease of reference, we will collectively define the term “click-stream” to encompass any user-initiated transition to a new content page, whether it is a mouse click, key depression or history state change.

Transition Sensor events are used to trigger the play of an advertisement only if, by then, all the media and player files for that advertisement have been fully cached into browser disk cache **430**. Otherwise, play of that advertisement is deferred until after all those files are cached and the advertisement is ready to be rendered and, importantly, in response to the next user-initiated transition.

Client browser **7** produces init (initialize) and start and stop Transition Sensor events, as symbolized by line **505** and **510**, respectively. The init and start events are produced by the browser to initialize (i.e., load and instantiate) and start the Transition Sensor applet. The stop events are also produced by the browser, though through a Transition Sensor stop method which has been substituted for a standard browser stop method, in response to detection, by the Transition Sensor, of user-initiated page transitions. These events control the state of applet **422**. Transition Sensor applet **422** communicates directly with AdController applet **424**, as symbolized by line **535**—such as to pass an Internet address of an advertising server, and indirectly, as symbolized by line **530**, through applet registry **426**. Registry **426** passes information, as symbolized by line **540**, to AdController applet **424**.

As noted above, AdController applet **424** extends standard Java applet class definitions by over-riding standard Java applet init, start, run, stop and destroy life cycle methods. Doing so, particularly in the case of the Stop



method (which will be described below in conjunction with FIG. 18), permits the AdController applet to persist in browser disk cache 430 as the user navigates across successive pages and web sites.

Advantageously, the AdController applet can readily function in a wide variety of environments, without changes to the coding of the applet itself. This is accomplished through downloading of an external configuration file (specifically file 620 shown in FIGS. 6A and 6B, which will be discussed below), as part of the applet files, from agent server 15. Suitably changing parameter values in the configuration file permits the behavior of applet 424 to be readily changed to suit a desired environment without a need to utilize a different version of that applet for each different environment, otherwise requiring different software classes and with attendant modifications and re-compilation.

Execution of AdController applet 424 begins by Transition Sensor applet 422 calling a standard init Applet method, which downloads the external configuration file, followed by extracting and saving its configuration parameters. These parameters are supplied, as symbolized by line 515, to the AdController applet, during its execution in order to define its behavior given its current execution environment.

As noted above, AdController applet 424 “politely” and transparently downloads advertising (including media and, where necessary, player) files, through browser 7 into browser disk cache 430, for each and every advertisement that is to be subsequently and interstitially played. A data path through which advertisements are downloaded is shown in FIG. 5 by dot-dashed lines; while that for advertisement play is shown in this figure by dotted lines.

Specifically, to download and play advertisements, applet 424 implements Ad Pipeline 545 (which will be discussed in detail below in conjunction with FIG. 14). Pipeline 545 implements various threads (processes) and data structures which collectively load advertising files into browser disk cache 430 (and, for media files, also into browser RAM cache) and then present fully downloaded advertisements. The pipeline implements Ad Producer, Ad Location and Ad Downloader processes (processes 1500, 1600, 1700 shown in FIGS. 15, 16 and 17, respectively, and discussed in detail below), and download queue 1430 and play queue 1470 (both of which are shown in FIG. 14 and discussed in detail below).

In essence, once Transition Sensor applet 422, as shown in FIG. 5, supplies AdController applet 424 with a URL of an AdDescriptor file, Ad Pipeline 545 then downloads, as symbolized by dot-dashed line 520, the AdDescriptor file, via agent server 15 (serving as a proxy server), from a remote advertising management system. As noted above, this file contains a manifest of media and player files necessary to fully play a complete advertisement. Once this AdDescriptor file has been downloaded into Ad Pipeline 545, pipeline 545 then “politely” downloads, as symbolized by line 525, each file specified in the manifest—to the extent that file does not already reside on the client hard disk. Pipeline 545 then, once the downloading (to the extent needed) is complete, writes the AdDescriptor file to the play queue and each downloaded file specified therein to browser disk cache 430; hence forming a queued advertisement for subsequent access.

At the inception of an interstitial interval, signaled by a Transition Sensor stop event, the AdController applet interstitially plays an advertisement that has then been completely queued—both in terms of its media and player files. In particular, at the start of that interval, the Ad Pipeline retrieves an AdDescriptor that is then situated at the head of

a play queue. Media players 565 required by that advertisement, as specified in the AdDescriptor file, are started in the order specified in that file along with their corresponding media file(s). A resulting processed media stream, produced by the player(s), and as symbolized by line 570, is rendered through browser 7 to the user. Media players 565 may permanently reside, i.e., apart from being downloaded by agent 420, on the client hard disk (thus being implemented by resident player files 410 as shown in FIG. 4) or be downloaded by pipeline 545 into browser disk cache 430 (and also browser RAM cache) for subsequent access and use (thus stored within files 437 shown in FIG. 4).

Once an advertisement completely plays, AdController applet 424, as shown in FIG. 5, establishes an appropriate log entry for a “user impression” for that advertisement. Advertisement files are retained in the browser disk cache until that cache completely fills, at which point these files, like any other content files stored in that cache, are deleted by the browser on a first-in first-out (i.e., age order) basis. Media players 565, browser 7 and browser disk cache 430 are all shown in dashed lines as these components, while being used by the AdController agent, are not viewed as constituting components solely within the agent itself.

FIGS. 6A and 6B collectively depict a high-level flow-chart of processing operations 600 performed by AdController agent 420; the correct alignment of the drawing sheets for these figures is shown in FIG. 6. Though, the operations depicted in this figure—and also in FIGS. 8, 9A and 9B, 12, and 14–19—occur through a multi-threaded approach to process multiple advertisements on a pipelined basis, to simplify all these figures, the sequential processing flow shown in each of these figures is that which processes a single common advertisement. Description of threads and classes is provided to the extent needed to provide a sufficient understanding to those skilled in the art as to how these sequential processing flows would preferably be implemented through a multi-threaded Java class methodology.

Upon entry into process 600 as shown in FIGS. 6A and 6B, which occurs in response to an Transition Sensor init event from browser 7, block 610 is performed. Through this block, Transition Sensor applet 422 instructs the applet registry to load the AdController applet. Once that occurs, block 615 is performed through which external AdController configuration file 620 is retrieved from agent server 15. Thereafter, through decision block 630, agent 420 waits, by looping through NO path 631, until browser 7 generates a Transition Sensor start event. When such an event occurs, execution proceeds, via YES path 633 emanating from this decision block, to block 635. Through this latter block, AdController applet 424 obtains an Internet address of an advertisement management system (e.g., system 25) from which the agent is to retrieve AdDescriptor file 645. Applet 424 then passes this address to Ad Pipeline 545. The Ad Pipeline, as indicated in block 640, then retrieves AdDescriptor file 645 from this address and particularly through agent server 15 serving as a proxy server. Once this file is retrieved, the agent performs block 650 which “politely” downloads all the media and player files 655 (to the extent each file does not already reside on the client hard disk), from advertising management system 25 (residing on advertising server 20), and, through block 660, stores these files into browser disk cache 430 (and, in the case of media files, into browser RAM cache). As noted above, these files are downloaded via agent server 15, which here too serves as a proxy server. This downloading continues until either it finishes or a Transition Sensor stop event generated by the browser arises, whichever occurs first. As to the stop event,

decision block 665 tests for its occurrence, with execution looping back, via NO path 666, in the absence of such an event. However, whenever this event occurs, such as (as discussed above) in response to a user-initiated page transition, decision block 665 routes execution, via YES path 668, to block 670. This latter block then, using media players 565, plays an advertisement then fully queued in the play queue on the browser disk cache, i.e., an AdDescriptor file for this ad then resides at a head of the play queue and all associated media and player files for that advertisement, as specified in that AdDescriptor file, then reside on the client hard disk.

3. AdController applet 424

FIG. 7 depicts a high-level block diagram of basic execution threads that implement AdController applet 424.

As shown, in response to a Transition Sensor init event produced by the client browser, one thread executes block 710 to initialize AdController applet 424. This block performs the downloading (to the extent necessary) and instantiation of applet 424. In response to a Transition Sensor start event produced by the client browser, another thread, by executing block 720, starts the AdController applet. Once this applet is started, this applet, in turn and as discussed above, through execution of block 730, enables downloading of advertising (media and player) files to commence. In response to a received Internet address of a remote ad management system (here, e.g., system 25 shown in FIGS. 1B and 1C) supplied by the Transition Sensor applet, a third thread requests, through execution block 740 shown in FIG. 7, an AdDescriptor file from the ad management system situated at this address and then downloads AdDescriptor file 645 received in response. If, by this time, block 730 has enabled advertisement downloading, then advertising files, as specified in AdDescriptor file 645, are "politely" downloaded as required. In response to a Transition Sensor stop event produced by the client browser and which signals an inception of an interstitial interval, another thread, here commencing with execution of block 750, suspends downloading of advertisement files in favor of displaying a queued advertisement. Once this downloading is suspended, this last thread invokes block 760 to commence play of an advertisement then situated, in terms of its AdDescriptor file, at a head of the play queue.

FIG. 8 depicts a high-level flowchart of processing operations 800 performed by AdController applet 424.

Upon entry into operations 800, which occurs in response to an init event produced by the Transition Sensor applet, block 805 is performed. Through this block, the AdController applet is initialized. This includes downloading files, to the extent needed, for this applet from the agent server and then instantiating this applet. Once this occurs, block 810 tests for an occurrence of AdController start event produced by the Transition Sensor applet. Until this event occurs, execution merely loops back, via NO path 812, to block 810. When this event occurs, decision block 810 routes execution, via YES path 814, to block 815. This latter block, retrieves external AdController configuration file 620 from the agent server. Thereafter, block 820 occurs through which the AdController applet creates and starts Ad Pipeline 545. Once the pipeline is fully started, then, block 825 is performed to enable advertisement files to be "politely" downloaded into the Ad Pipeline and to thereafter actually download such files. While advertisement files are being downloaded or thereafter, if such downloading has completed, decision block 830 tests for an occurrence of a Play Ad event. If no such event occurs, then execution loops back, via NO path 833, to decision block 830 to continue any

further downloading. If however, a Play Ad event occurs, then decision block 830 routes execution, via YES path 837, to block 840. This latter block suspends further downloading of advertisement files into the Ad Pipeline. Once this occurs, then block 845, when performed, issues a request to the Ad Pipeline to play an advertisement having its AdDescriptor file then located at the head of the play queue. While the advertisement is being played, decision block 850 tests for an occurrence of a shutdown event generated by the browser, such as caused by, e.g., a user-initiated transition or the user closing an advertisement window or closing the browser itself. If such an event does not occur, decision block 850 routes execution, via NO path 853, back to block 825 to re-enable "polite" download of advertisement files once again. If such a shutdown event occurs, then processing operations 800 terminate, via YES path 857.

FIGS. 9A and 9B collectively depict a flowchart of processing operations 900 performed by AdController applet 424 specifically for processing an advertisement; the correct alignment of the drawing sheets for these figures is shown in FIG. 9.

Upon entry into operations 900, block 905 is performed to receive a request, issued by the Transition Sensor applet, to download a next advertisement, specifically a corresponding AdDescriptor file. This request contains an Internet address of a remote ad management system. In response to this request, AdController applet 424 performs block 910 to request Ad Producer process (also being a thread) 1500 to download an ad. The Ad Producer process, as will be discussed below in conjunction with FIG. 15, requests advertisement files, specifically an AdDescriptor file, from an Internet address communicated by the Transition Sensor applet. Thereafter, through block 915, the Ad Producer process blocks (i.e., it actively waits for its input data) until this process receives the Internet address of the remote advertising management system. Thereafter, block 920 executes to cause Ad Location process (also being a thread) 1600 to block until such time as the AdDescriptor file is fully downloaded by Ad Producer process 1500 and is provided to the Ad Location process. Ad Location process 1600, as will be discussed below in conjunction with FIG. 16, performs the following tasks: (a) on startup of process 1600, this process creates an Ad Producer object; (b) it asks Ad Producer process 1500 for next AdDescriptor file 645; and (c) once process 1600 obtains such AdDescriptor file 645 and if download queue 1430 (see FIG. 14) is not full, it writes that file into this queue. If this queue is then full, process 1600 simply waits until the queue is not full before writing the AdDescriptor file into the queue. Once the AdDescriptor file has been completely downloaded, Ad Location process 1600 inserts, as shown in block 925, this file into download queue 1430.

Once AdDescriptor file 645 is inserted into the download queue, then Ad Downloader process (also being a thread) 1700 executes. This process, as will be discussed below in conjunction with FIG. 17, performs a single chain of tasks.

First, as shown by block 930, process 1700 blocks until such time as the AdDescriptor file for the advertisement to then be downloaded becomes available in the download queue. During its execution, this process asks download queue 1430 if there is an AdDescriptor file therein, i.e., such a file for which advertising files need to be downloaded. If the download queue is empty, then AdDescriptor process 1700 both waits until that queue is not empty and also retrieves the AdDescriptor file over the network. Once the Ad Downloader process has retrieved the AdDescriptor file, this process, through execution of block 935, requests



browser cache proxy 1450 (shown in FIG. 14) to download all ad media and player files. In response to this request, browser cache proxy 1450 then downloads, as shown by block 940, all the advertising files specified in the AdDescriptor file, into browser disk cache (and, in the case of media files, into browser RAM cache). Once all the advertising files have finished downloading, the Ad Downloader process, as shown in block 950, moves the AdDescriptor file to play queue 1470 (see FIG. 14). However, if the play queue is then full, the Ad Downloader process will wait until the play queue is not full before moving the AdDescriptor file into this queue.

The Browser Cache Proxy implements an interface to an abstract cache. The cache implementation could be any kind of cache—the browser disk or RAM cache, a Java virtual memory cache, a local raw disk cache, and so forth. Once passed through this cache proxy, the media files that constitute an advertisement will have been downloaded into both disk and RAM cache of the browser. Whenever the Ad Downloader process subsequently tries to access any media file having an identical URL to that downloaded, this process will first attempt to load the files from the browser disk cache or browser RAM cache instead of downloading the file, via the Internet, from its advertising management server; thus leveraging, even across different referring web pages or sites and to the extent possible, a one time download of an advertising file across different advertisements.

Next, should a Transition Sensor stop event occur, i.e., indicative of a start of a next interstitial interval, then Transition Sensor stop method 1800 will request, as shown by execution of block 955, that AdController applet 424 then play an advertisement. In response to this request, an event scheduler thread within the applet will block, as shown in block 960, until such time as applet 424 responds to this request by initiating play of an advertisement. The event scheduler thread controls playing of advertisements to the user. This thread determines when to execute media players specific to the next advertisement in the play queue (i.e., in terms of corresponding AdDescriptor files situated in that queue), as well as provides a callback method which the player executes when that player has successfully completed presenting an advertisement as specified in its corresponding AdDescriptor file. Once the AdController applet has initiated play of an advertisement, then, as shown by block 965, the event scheduler retrieves an advertisement, specifically the corresponding AdDescriptor file, then situated at the head of the play queue. Thereafter, the event scheduler, as shown in block 970, launches execution of the specific media player(s) 565 (see FIG. 5), as specified in the corresponding AdDescriptor file, to play this particular advertisement. The browser disk cache provides the associated content files for this advertisement to the media player(s). Once the advertisement has been fully presented, then, as shown in block 975, AdController applet 424, through the event scheduler thread, appropriately logs this presentation into a log file maintained in the browser disk cache for subsequent uploading to the agent server. Execution then exits from operations 900.

A logger process (also implemented as a thread) keeps track of all log entries that need to be sent back to the agent server. This process simply timestamps entries and adds them to a log buffer. Then, periodically, the logger process will flush the log back to the agent server where those entries can be archived and analyzed.

For an advertisement, player mechanisms take associated media files specified in the associated AdDescriptor file from the browser cache and actually display these files to a user

via a viewable frame or window. The user will view a pre-cached smoothly playing advertisement out of the browser disk cache and, where appropriate for media files, from browser RAM cache, rather than being streamed in over the Internet. Four modes for displaying advertisements are supported; namely, user-event triggered ad play, frame-targeted ad play, timer-based ad play and PopUp Java frame play. Each of these player mechanisms uses a media player module (contained within media players 565 shown in FIG. 5) and a player thread. The player thread provides an actual presentation of advertising media to the user then operating the client browser. The combination of a player and a player thread provides capabilities of: controlling time-based frequency of advertisement play using an agent configurable timer; displaying advertising media files in a browser window or Java frame; waiting a configurable amount of time (usually the length of the advertisement as specified in its AdDescriptor file); and terminating the advertisement visually upon completion, or at a request of the user if the advertisement, as configured in its AdDescriptor file, permits pre-mature termination.

A frame-targeted play renders advertisement media onto a browser window. Such play is interruptible and restartable upon user-demand. Timer-based ad play utilizes a separate thread that continuously loops to: obtain an AdDescriptor file from the play queue; display that advertisement using a player and player thread; and sleep for a specified amount of time before repeating this sequence. Timer-based ad play is also interruptible and restartable upon user-demand. The result of this type of advertisement play is that the user will periodically view advertisements delivered at regular time intervals rather than by user initiated events. The PopUp Java frame play is a separate thread that also continuously loops to: obtain an AdDescriptor file from the play queue; waits for a signal that a user-initiated transition is occurring; pops up a display window (“pop-up” window) in the browser, for a pre-defined period of time, and presents the advertisement in that window; and removes the pop up window before repeating this sequence. The result of the PopUp Java Player is that the user will view successive advertisements each for pre-defined time interval (which can vary from one advertisement to the next, as specified in the AdDescriptor files for each such advertisement) whenever the user transitions between one web page and the next. Once an advertisement is completely played and in the absence, as discussed above of any instructions in the AdDescriptor file to replay that advertisement, such as through, e.g., timer-based ad play, the associated AdDescriptor file is effectively “pulled off” the play queue.

In particular, downloading of advertisement files occurs, as discussed previously, continuously as effectively a background process, using a separate asynchronous thread. The stop method of the Transition Sensor (specifically Transition Sensor stop method 1800 as will be described below in conjunction with FIG. 18) is responsible for generating a play event to the AdController agent. This event notifies the agent of an opportunity to present a downloaded advertisement to the user. This stop method is called automatically by the client browser whenever a user transitions off a web page that contains the embedded advertising tag. In particular, this method invokes a start player method in the AdController agent. The start player method, in turn, invokes a similarly named method, in the event scheduler, which initiates and controls the presentation of advertisements during content page transitions. The event scheduler ensures all media files for an advertisement have been transparently downloaded before their presentation, as well as exercises control over

actual execution of the appropriate player classes required to visibly render the advertisement. In that regard, the event scheduler instantiates and invokes a player class appropriate for the current advertisement by calling a start method of that class. This start method creates the player thread that performs visual rendering of the advertisement. Then, this start method calls a run method of the player thread in order to visually present the advertising media from the browser disk and RAM caches. Upon completion, based on the configuration of the advertisement, the run method, by executing its own stop method, terminates the advertisement either upon detecting a close request by the user or completion of ad play timeout. The stop method performs any player software termination and cleanup, finally executing a callback to the scheduler object.

4. Inter-applet events involving Transition Sensor applet 422

FIG. 10 depicts inter-applet events 1000 that occur within AdController agent 420 during execution of Transition Sensor applet 422.

As shown and discussed above, whenever a browser interprets and then executes advertising tag 40, specifically tag 42 therein, situated within content page 35, this causes, as symbolized by line 1010, the browser to download script 200 (see FIGS. 2A and 2B) from the agent server. This applet, in turn, dynamically writes Transition Sensor applet 210 (also referred to as applet 422) into the referring web content page. As discussed above, once this applet is instantiated and executed by the client browser, the applet, in turn, instantiates applet registry 426.

Once the applet registry is instantiated, the Transition Sensor queries the registry, this operation being symbolized by line 1015, to determine current status of the AdController applet. If, as symbolized by line 1020, the registry indicates that the AdController applet is not loaded and hence is not executing, then Transition Sensor applet 422 loads, as symbolized by line 1025, AdController applet 424 from the browser disk cache, and then instantiates and starts this applet. Once the AdController applet is instantiated, the Transition Sensor applet writes, as symbolized by line 1030, appropriate entries, indicating that both the Transition Sensor applet is loaded and, as symbolized by line 1035, that the AdController applet is loaded, into the applet registry. Once this occurs, then the applet registry returns, as symbolized by line 1040, an appropriate handle for the AdController applet to the Transition Sensor in order to permit the latter to refer to the former. Thereafter, as symbolized by line 1060, the Transition Sensor passes, as discussed above, a request containing an Internet address of an advertisement management system to the AdController applet to download an AdDescriptor file, for an advertisement, from that address. This address is specified in tag 44 of advertising tag 40 and, as symbolized by dashed line 1050, incorporated into the request. Thereafter, the Transition Sensor, in response to a user-initiated transition (click-stream) to a next content web page, executes its stop method (method 1800 shown in FIG. 18) to instruct, i.e., issue a request to, as symbolized by line 1065, the AdController applet to play a fully downloaded advertisement having its corresponding AdDescriptor file then situated at the head of the play queue. Once this occurs, the Transition Sensor applet terminates its execution until the browser next encounters, interprets and executes a content page containing advertising tag 40 at which point the Transition Sensor applet is re-loaded and re-started; and so forth.

5. Transition Sensor applet 422

FIG. 11 depicts a high-level block diagram of basic processing threads that implement Transition Sensor applet 422.

As shown, in response to a Init (Initialize) Transition Sensor applet event produced by the client browser, a thread commences by executing block 1110 to initialize Transition Sensor applet 422. This thread, in turn, executes block 1120 to load AdController applet 424 from browser disk cache or download it from the agent server, if necessary, and then load it. Thereafter, this thread executes block 1130 to obtain the Internet address of an advertising management system (e.g., system 25 shown in FIGS. 1B and 1C, 2A and 2B, and 10) in tag 44 from advertising tag 40.

As shown in FIG. 11, in response to a Start Transition Sensor applet event generated by the client browser, another thread commences by executing block 1140 to enable Ad Downloader process 1700 (as discussed above, and to be discussed in detail below in conjunction with FIG. 17) to commence "polite" downloading an AdDescriptor file and all required and associated advertisement files (both media and player) into the browser disk cache.

Further, as shown in FIG. 11, in response to a Stop Transition Sensor applet event generated by the client browser, a third thread commences by executing block 1150 to disable Ad Downloader process 1700 and thus suspend further downloading of advertisement files. Once this occurs, this thread then executes block 1160 to instruct the AdController applet to play a fully downloaded advertisement having its corresponding AdDescriptor file then situated at the head of the play queue.

FIG. 12 depicts a high-level flowchart of processing operations 1200 performed by Transition Sensor applet 422.

Upon entry in operations 1200, decision block 1210 tests for an occurrence of an init event produced by the client browser. Until such an event occurs, execution loops back, via NO path 1213, to block 1210. When this event occurs, execution proceeds, via YES path 1217 to block 1220 which, when performed, initializes Transition Sensor applet 422. Thereafter, block 1230 is performed through which the Transition Sensor applet 424 instructs, by issuing a request to, the AdController applet to download an advertisement, specifically as discussed above an AdDescriptor file from an ad management server specified in the advertising tag. Once this occurs, decision block 1240 tests for an occurrence of a Transition Sensor start event generated by the client browser. Until such an event occurs, execution loops back, via NO path 1243, to block 1240. When this particular event occurs, execution proceeds, via YES path 1247 to block 1250 which, when performed, enables Ad Pipeline 545 to download the AdDescriptor file and associated advertising files.

Next, decision block 1260 tests for an occurrence of a Transition Sensor stop event generated by the client browser. Until such an event occurs, execution loops back, via NO path 1263, to block 1260. When a Transition Sensor stop event occurs, execution then proceeds, via YES path 1267 to block 1270 which, when performed, requests that AdController applet 424, specifically via Ad Pipeline 545, then play an advertisement.

6. Ad Loader process 1300

FIG. 13 depicts a high-level block diagram of Ad Loader process 1300 which forms a portion of AdController applet 424. Process 1300 provides an advertiser (specifically an advertising programmer) with control over various functions, for advertisement play and logging, implemented by the AdController applet, specifically how and where this applet retrieves advertisements across a networked connection and how those advertisements are played. Through use of the Ad Loader, the AdController applet can be controlled, to an extent desired, by external programmatic calls.

As shown, this process includes Ad Loader API (application programming interface) 1310 which interfaces to Ad Pipeline 545 and through this pipeline controls how advertisements are presented, as symbolized by block 1370, by the player mechanisms. In particular, the Ad Loader API provides information regarding and, through setting various program variables, permits programmer control over advertisement display and downloading operations. In that regard, these variables provide a callback to the AdController applet indicating when a content page to which the user has just transitioned has completed its downloading; and can be used to: instruct the AdController applet when to download a next advertisement, when to play a next advertisement fully queued in the Ad Pipeline, start and stop a play timer (for use with, e.g., timer-based ad play, as discussed above), log a message, set a mode so as to specify a desired location to display advertisements, suspend and resume download of advertisement files into the Ad Pipeline, suspend a current download for a given period of time, and suspend and resume advertisement play by the player mechanisms.

In that regard, the Ad Loader API configures Ad Pipeline 545 such that AdDescriptor file 645 is downloaded, as symbolized by block 1320, from a remote ad management system into the Ad Pipeline in response to receipt of an Internet address of an ad management system and, for targeted advertisements, a URL of a referring web page address. As symbolized by block 1330, the API configures the Ad Pipeline such that advertisement downloading is enabled only when AdController applet 424 is not playing an advertisement. Furthermore, as symbolized by block 1340, the API configures the Ad Pipeline such that advertisement downloading is disabled whenever the AdController applet is playing an advertisement. Furthermore, as symbolized by block 1350, the API configures the Ad Pipeline such that advertisement play is to commence in response to a request to play a next advertisement, i.e., one that is fully cached in the browser disk cache and having its AdDescriptor file then situated at the head of the play queue.

7. Ad Pipeline 545

FIG. 14 depicts a high-level block diagram of Ad Pipeline 545. As discussed above, the Ad Pipeline implements various threads and data structures which collectively load advertising files (needed media and player files) into the browser disk cache and, for media files, also into browser RAM cache, and then present fully downloaded advertisements. As noted, the Ad Pipeline employs Ad Producer process 1500, Ad Location process 1600 and Ad Downloader process 1700 (all of these processes, as noted above, are also threads).

In response to an incoming request to download an advertisement, Ad Pipeline 545 is invoked. Specifically, within this pipeline, first block 1410 executes to invoke Ad Producer process 1500 in response to an incoming request to download an advertisement. As discussed above, this request, issued by the Transition Sensor applet, includes an Internet address of a remote ad management system (e.g., system 25 shown in FIGS. 1B and 1C) on which an advertisement resides and is to be downloaded (through agent server 15 as a proxy server). Ad Producer process 1500, as will be discussed below in conjunction with FIG. 15, requests advertisement files, specifically an AdDescriptor file (e.g., file 645), from an Internet address specified in the request. During its execution, the Ad Producer process waits until it receives the Internet address of the remote advertising management system, whereupon this process then downloads AdDescriptor file 645 from the specified ad management system. Once this file has been downloaded,

block 1420, shown in FIG. 14, executes to invoke Ad Location process 1600 (which will be discussed in detail below in conjunction with FIG. 16). During its execution, Ad Location process 1600 blocks until such time as AdDescriptor file 645 is fully downloaded by Ad Producer process 1500 and is provided to the Ad Location process, whereupon the Ad Location process writes this AdDescriptor file into download queue 1430.

After AdDescriptor file 645 has been written into the download queue, Ad Location process 1600, as will be discussed below in conjunction with FIG. 16, performs the following tasks: (a) on startup of process 1600, this process creates an Ad Producer object; (b) this process asks Ad Producer process 1500 for next AdDescriptor file 645; and (c) once process 1600 obtains AdDescriptor file 645 and, if download queue 1430 is not full, process 1600 writes that file into this queue. If this queue is then full, process 1600 simply waits until the queue is not full before writing the AdDescriptor file into the queue. Once the AdDescriptor file has been completely downloaded, Ad Location process 1600 inserts, as shown in block 925, this file into download queue 1430.

Once AdDescriptor file 645 is inserted into the download queue, then block 1440 executes to invoke Ad Downloader process 1700. Process 1700, which will be discussed below in conjunction with FIG. 17, performs a single chain of tasks. First, process 1700 blocks until such time as the downloaded AdDescriptor file has become available in the download queue. During its execution, this process asks download queue 1430 if it contains an AdDescriptor file, e.g., file 645. If so, then advertising files need to be downloaded for that particular AdDescriptor file. If the download queue is empty, then process 1700 both waits until that queue is not empty and also retrieves the AdDescriptor file over the network. Once Ad Downloader process 1700 has obtained this AdDescriptor file, process 1700 then downloads, all the media and required player files specified in the AdDescriptor file by using Browser Cache Proxy 1450, into browser disk (and RAM) cache 1460. Once all the advertising files have finished downloading, process 1700 moves the AdDescriptor file to play queue 1470. However, if the play queue is then full, the Ad Downloader process waits until play queue 1470 is not full before moving the AdDescriptor file into this queue for subsequent ad play. As discussed above, AdDescriptor file 645 for a fully queued ad (i.e., with its all the associated media and player residing on the client hard disk) is subsequently retrieved from play queue 1470 in response to a request to play an advertisement, this request being issued in response to a Transition Sensor stop event.

8. Ad Producer process 1500

FIG. 15 depicts a high-level block diagram of Ad Producer process 1500. As noted above, this process requests an AdDescriptor file from an Internet address communicated by the Transition Sensor applet and subsequently downloads that file into the browser disk cache.

As shown, upon entry into process 1500, execution first proceeds to decision block 1510. This block determines whether a URL has been received, from the Transition Sensor, from which to fetch an AdDescriptor file. If such a URL has not yet been received, then execution loops back, via NO path 1517, to this decision block. Alternatively, if such a URL has been received, then execution proceeds, via YES path 1513, to block 1520 which, in turn, stores this URL, as Ad URL 1530, for use during a next successive advertisement download opportunity.

Once this URL has been so stored, execution proceeds to decision block 1540. This block tests for an occurrence of a



user-initiated event (click-stream) signifying that advertisement downloading can now occur, such as, e.g., when the user has just closed an existing advertisement frame and a next successive content page to which the user has transitioned is being rendered by the client browser. If such an event has not yet occurred, e.g., the next successive content web page is downloading, then execution merely loops back, via NO path 1543, back to decision block 1540. However, if such an event occurs, then this decision block routes execution, via YES path 1547, to block 1550. This latter block, when executed, downloads AdDescriptor file 645 using the URL communicated by the Transition Sensor. Once this file is completely downloaded, then block 1560 executes to transfer this file to Ad Location process 1600. Thereafter, execution loops back, via path 1565, to decision block 1510, and so forth.

9. Ad Location process 1600

FIG. 16 depicts a high-level block diagram of Ad Location process 1600. This process, as discussed above, accomplishes the following tasks: (a) on startup of this process, process 1600 creates an Ad Producer object; (b) process 1600 asks Ad Producer process 1500 for next AdDescriptor file 645; and (c) once process 1600 obtains AdDescriptor file 645 and, if download queue 1430 (see FIG. 14) is not full, process 1600 then writes that file into this queue. If this queue is then full, process 1600 simply waits until the queue is not full before writing the AdDescriptor file into the queue.

Upon entry into process 1600 and with respect to advertisement downloading itself, execution proceeds to decision block 1610. This decision block, when executed, determines whether an Internet address (URL) of an ad management system has been received from the Transition Sensor applet for a next successive advertisement download. If that address has not yet been received, then execution merely loops back, via NO path 1613, to decision block 1610. Alternatively, if such an address has been received but not yet processed, then decision block 1610 routes execution, via YES path 1617, to block 1620. This latter block requests Ad Producer process 1500 to download an AdDescriptor file, e.g., file 645, from this URL. Once this request occurs, execution proceeds to decision block 1630 to determine whether this AdDescriptor file has been completely downloaded. If this file download is still occurring, then execution merely loops back, via NO path 1633, to block 1630 to await completion of the download. Once this download completes, decision block 1630 routes execution, via YES path 1637, to block 1640. This latter block writes the downloaded AdDescriptor file into download queue 1430 (providing this queue is not full). Once this occurs, execution is directed, via path 1645, back to decision block 1610, and so forth.

10. Ad Downloader process 1700

FIG. 17 depicts a high-level block diagram of Ad Downloader process 1700. Essentially, as discussed above, process 1700 determines, from the download queue, if that queue contains an AdDescriptor file, e.g., file 645. If it does contain such an AdDescriptor file, then advertising files need to be downloaded for that file. Consequently, process 1700 then downloads required advertising files specified in that AdDescriptor file. Once this fully occurs, process 1700 moves the AdDescriptor file to the play queue.

In particular upon entry into process 1700, execution proceeds to decision block 1710. This decision block determines whether the download queue then contains an AdDescriptor file, e.g., file 645. If the queue is empty, then execution merely loops back, via NO path 1717, to this decision block to await such an AdDescriptor file. However,

if download queue 1430 then contains such a file, process 1720 obtains the AdDescriptor file then situated at the head of this queue. Thereafter, block 1730 executes. This block downloads all the required advertising files, not then resident on the client hard disk, into browser proxy cache 1450. This block also transfers all the associated media files in the browser proxy cache to the browser RAM cache. Execution then proceeds to decision block 1740 which determines whether all required advertising files have then been downloaded. If any such file remains to be downloaded, then decision block 1740 routes execution, via NO path 1747, back to block 1730 to download that file. Alternatively, if all the required advertising files have been downloaded, then execution proceeds, via YES path 1743, to block 1750. This latter block moves the AdDescriptor file from download queue 1430 to an end of play queue 1470. Once the AdDescriptor file is written into the play queue, the corresponding advertisement is then ready to be presented to the user, in order relative to other AdDescriptor files then queued in the play queue, during an ensuing interstitial interval.

11. Transition Sensor stop method 1800

FIG. 18 depicts a flowchart of stop method 1800 invoked by Transition Sensor applet 422. This method, in response to a stop event generated by the browser, suspends downloading of advertisement files and initiates interstitial ad play.

In particular, upon entry into method 1800, decision block 1810 executes to determine if a stop event has been received from browser 7. If such a stop event has yet not occurred, then execution loops back, via NO path 1813, back to block 1810 to await occurrence of this event. When this event occurs, decision block 1810 directs execution, via YES path 1817, to decision block 1820. This latter decision block determines if AdController applet 424 is then loaded and executing. If this applet is not then executing, decision block 1820 routes execution, via NO path 1827, to block 1830. This latter block inhibits any request from being made to the AdController applet to play any advertisement until that applet is executing and, once that occurs, a next user-initiated (click-stream) event occurs. Thereafter, execution of method 1800 terminates. Alternatively, if the AdController applet is loaded and executing, then decision block 1820 routes execution, via YES path 1823, to block 1840. This latter block requests the AdController applet to play a next advertisement. Once this request is issued, then execution proceeds to block 1850. This block, in turn, requests the AdController applet to suspend "polite" background downloading of advertisement files while a next successive web content page, as requested by the user, is being downloaded by the browser. Once block 1850 executes, execution of method 1800 terminates.

12. Transition Sensor start method 1900

FIG. 19 depicts a flowchart of start method 1900 invoked by Transition Sensor applet 422. This method, in response to a start event generated by the browser, resumes background downloading of advertisement files.

Specifically, upon entry into method 1900, execution proceeds to decision block 1910 which, when executed, determines if a start event has been received from browser 7. If such a start event has not yet occurred, then execution loops back, via NO path 1913, back to block 1910 to await occurrence of this event. When this event occurs, decision block 1910 directs execution, via YES path 1917, to decision block 1920. This latter decision block determines if AdController applet 424 is then loaded and executing. If this applet is not then executing, decision block 1920 routes execution, via NO path 1927, to block 1930. Block 1930 inhibits any

request from being made to the AdController applet to download any advertisement until that applet is executing and, once that occurs, a next user-initiated (click-stream) event occurs. Once the AdController applet begins executing and thereafter a next user-initiated (click-stream) event occurs, execution proceeds to block 1940. This latter block requests the AdController applet to resume background downloading of advertisement files. Once this downloading is resumed, method 1900, through execution of block 1960, waits for browser 7 to call Transition Sensor stop method 1800 whenever the user next unloads a web page currently rendered by the browser, i.e., causes a user initiated-event to transition to a next successive web page. Alternatively, if the AdController applet is loaded and executing, then decision block 1920 routes execution, via YES path 1923, to block 1950. Since at this point the next successive content web page has been fully executed by the browser and is, e.g., rendered to the user, block 1950 issues a request, through the applet registry, to the AdController applet to enable it to resume background downloading of advertisement files. Once this occurs, block 1940 is executed to issue a request to the AdController applet to resume the background downloading. Execution then proceeds to block 1960 to wait for browser 7 to call Transition Sensor stop method 1800 whenever the user next unloads a web page currently rendered by the browser, i.e., causes a user initiated-event to transition to a next successive web page. Whenever the browser generates a next Transition Sensor stop event, process 1900 terminates.

Although a single embodiment which incorporates the teachings of our present invention has been shown and described in considerable detail herein, those skilled in the art can readily devise many other embodiments and applications of the present invention that still utilize these teachings.

We claim:

1. Apparatus for use in rendering an information object in response to a first web page containing an embedded code, the apparatus comprising:

- a processor;
- a memory connected to the processor and storing both computer executable instructions and the first web page, the first web page having a plurality of computer readable instructions representing page content and the embedded code; and

an output device operative in conjunction with the processor;

wherein the processor, in response to the executable instructions and as a result of executing the code through a web browser, downloads an agent, from a first server, into the memory and subsequently executes the agent under control of the browser, wherein the agent:

downloads, from a second server and while the computer renders the first web page to a user through the output device, at least one file which is to be subsequently employed, by the processor, to render an information object;

monitors a click-stream produced by the user to detect a user navigation event signifying a user action to transition from the first web page to a next successive web page and which signifies a start of a next interstitial interval; and

in response to the user navigation event, suspends further downloading of files and processes the one file so as to render the information object through the output device to the user during the interval.

2. The apparatus in claim 1 wherein the information object comprises a web advertisement, the code comprises advertising code and the one file comprises an advertisement file.

3. The apparatus in claim 2 wherein the user navigation event comprises an affirmative action taken by the user, through the browser, to navigate from the first web page to the next successive web page, wherein the action comprises a mouse click, a key depression or a user-invoked state change in a stored history of web pages previously visited by the user.

4. The apparatus in claim 3 wherein the advertisement file comprises an Ad Descriptor file or at least one advertising file specified in the Ad Descriptor file, the advertising file being either a media file or a player file.

5. The apparatus in claim 4 wherein the processor, in response to execution of the agent, overrides default life cycle methods defined in the browser with corresponding substitute methods such that the agent persistently remains in browser storage as the browser transitions across successive web pages and different web sites.

6. The apparatus in claim 5 wherein the life cycle methods comprise at least one of start, run, stop, initialize and destroy methods.

7. The apparatus in claim 6 wherein the agent comprises a Transition Sensor applet and an Ad Controller applet, and the processor, during execution of the Transition Sensor:

instantiates and starts execution of the Ad Controller applet; and

monitors the click-stream so as to detect the user-initiated event such that the processor:

instructs the Ad Controller applet to download the Ad Descriptor file for the web advertisement from the second server into the browser storage on the computer; and

in response to an occurrence of the event, instructs the Ad Controller applet to cease any download of a further advertisement file specified in the Ad Descriptor file, to the extent any downloading of said further advertisement file is then occurring, and initiates processing through the browser, of files for an advertisement that has been previously downloaded and is currently ready to be rendered so as to render the previously downloaded advertisement during the next interstitial interval to the user.

8. The apparatus in claim 7 wherein the processor in response to executing the advertising code:

determines, through the agent, whether a new version of either the Transition Sensor applet or the Ad Controller applet then resides on the distribution server relative to a corresponding version, if any, of the Transition Sensor and Ad Controller applets, respectively, then residing in the browser storage; and

if said new version exists on the distribution server, downloads the new version from the distribution server into the browser storage and executes the new version in lieu of the corresponding version.

9. The apparatus in claim 8 wherein the Ad Controller applet comprises a play queue, wherein the processor during execution of the Ad Controller applet:

once all the advertising files specified in an associated Ad Descriptor file for a corresponding advertisement reside in the browser storage on the computer, inserts the associated Ad Descriptor file into an end of the play queue; and

in response to the user navigation event and during the ensuing interstitial interval, processes advertising files

41

specified in a specific Ad Descriptor file then situated at a head of the play queue so as to render, through the output device, an advertisement, corresponding to the specific Ad Descriptor file, to the user.

10. The apparatus in claim 9 wherein the agent, in response to the occurrence of the user navigation event, generates a stop event which, when processed by the agent, suspends the downloading of further advertisement files and initiates processing of files specified in the Ad Descriptor file, then situated at the head of the play queue, so as to render the web advertisement associated therewith during the ensuing interstitial interval.

11. The apparatus in claim 10 wherein the Transition Sensor applet monitors user click stream so as to detect the user navigation event and, in response thereto, produce the stop event, and wherein Ad Controller applet processes the stop event to suspend said downloading of further advertisement files and to render the advertisement associated with the Ad Descriptor file then situated at the head of the play queue.

12. The apparatus in claim 11 wherein the output device is a display.

13. The apparatus in claim 11 wherein the first and second servers are a distribution server and an advertising file server, respectively.

14. The apparatus in claim 4 wherein the advertising code further comprises a component specifying the advertising server.

15. The apparatus in claim 14 wherein the processor, during execution of the Ad Controller applet and in response to the component contained in the code, downloads the Ad Descriptor file originating from the advertising server specified in the second component.

16. The apparatus in claim 4 wherein the Ad Descriptor file comprises a manifest of names of a plurality of predefined advertising files and associated configuration information necessary to properly play the downloaded advertisement through the browser.

17. The apparatus in claim 16 wherein the Ad Descriptor file comprises a list having: a name of each player and media file that constitutes the downloaded advertisement, a corresponding network address at which said each file can be accessed, configuration information for at least one of the player files for properly configuring the corresponding player to render an associated media file.

18. The apparatus in claim 9 wherein, if the Ad Controller applet is not executing at the occurrence of the user navigation event, the processor, in response to the stored instructions, processes the one advertisement file, so as to render the web advertisement, only after both the Ad Controller applet has started execution and a next successive user navigation event has occurred.

19. The apparatus in claim 9 wherein the output device is a display.

20. The apparatus in claim 9 wherein the first and second servers are a distribution server and an advertising file server, respectively.

21. The apparatus in claim 3 wherein the advertising code comprises an advertising tag and the processor, in response to execution of the tag:

dynamically writes a plurality of predefined applet tags that collectively implement a script into the first web page; and

downloads, in response to subsequent execution of the script, the agent from the first server into the memory and thereafter instantiates and executes the agent.

22. The apparatus in claim 21 wherein the advertisement file comprises an Ad Descriptor file or at least one adver-

42

tising file specified in the Ad Descriptor file, the advertising file being either a media file or a player file.

23. The apparatus in claim 22 wherein the processor, in response to execution of the agent, overrides default life cycle methods defined in the browser with corresponding substitute methods such that the agent persistently remains in browser storage as the browser transitions across successive web pages and different web sites.

24. The apparatus in claim 23 wherein the life cycle methods comprise at least one of start, run, stop, initialize and destroy methods.

25. The apparatus in claim 24 wherein the agent comprises a Transition Sensor applet and an Ad Controller applet, and the processor, during execution of the Transition Sensor:

instantiates and starts execution of the Ad Controller applet; and

monitors the click-stream so as to detect the user-initiated event such that the processor:

instructs the Ad Controller applet to download the Ad Descriptor file for the web advertisement from the second server into the browser storage on the computer; and

in response to an occurrence of the event, instructs the Ad Controller applet to cease any download of a further advertisement file specified in the Ad Descriptor file, to the extent any downloading of said further advertisement file is then occurring, and initiates processing through the browser, of files for an advertisement that has been previously downloaded and is currently ready to be rendered so as to render the previously downloaded advertisement during the next interstitial interval to the user.

26. The apparatus in claim 25 wherein the processor in response to executing the tag:

determines, through the agent, whether a new version of either the Transition Sensor applet or the Ad Controller applet then resides on the distribution server relative to a corresponding version, if any, of the Transition Sensor and Ad Controller applets, respectively, then residing in the browser storage; and

if said new version exists on the distribution server, downloads the new version from the distribution server into the browser storage and executes the new version in lieu of the corresponding version.

27. The apparatus in claim 26 wherein the Ad Controller applet comprises a play queue, wherein, the processor during execution of the Ad Controller applet:

once all the advertising files specified in an associated Ad Descriptor file for a corresponding advertisement reside in the browser storage on the computer, inserts the associated Ad Descriptor file into an end of the play queue; and

in response to the user navigation event and during the ensuing interstitial interval, processes advertising files specified in a specific Ad Descriptor file then situated at a head of the play queue so as to render, through the output device, an advertisement, corresponding to the specific Ad Descriptor file, to the user.

28. The apparatus in claim 27 wherein the agent, in response to the occurrence of the user navigation event, generates a stop event which, when processed by the agent, suspends the downloading of further advertisement files and initiates processing of files specified in the Ad Descriptor file, then situated at the head of the play queue, so as to render the web advertisement associated therewith during the ensuing interstitial interval.



43

29. The apparatus in claim 28 wherein the Transition Sensor applet monitors user click stream so as to detect the user navigation event and, in response thereto, produce the stop event, and wherein Ad Controller applet processes the stop event to suspend said downloading of further advertisement files and to render the advertisement associated with the Ad Descriptor file then situated at the head of the play queue.

30. The apparatus in claim 29 wherein the output device is a display.

31. The apparatus in claim 29 wherein the first and second servers are a distribution server and an advertising file server, respectively.

32. The apparatus in claim 21 wherein the advertising tag further comprises first and second components, the first and second components specifying the script and the advertising server, respectively.

33. The apparatus in claim 32 wherein the processor, during execution of the Ad Controller applet and in response to the second component contained in the tag, downloads the Ad Descriptor file originating from the advertising server specified in the second component.

34. The apparatus in claim 22 wherein the Ad Descriptor file comprises a manifest of names of a plurality of pre-defined advertising files and associated configuration information necessary to properly play the downloaded advertisement through the browser.

35. The apparatus in claim 34 wherein the Ad Descriptor file comprises a list having: a name of each player and media file that constitutes the downloaded advertisement, a corresponding network address at which said each file can be accessed, configuration information for at least one of the player files for properly configuring the corresponding player to render an associated media file.

36. The apparatus in claim 25 wherein, if the Ad Controller applet is not executing at the occurrence of the user navigation event, the processor, in response to the stored instructions, processes the one advertisement file, so as to render the web advertisement, only after both the Ad Controller applet has started execution and a next successive user navigation event has occurred.

37. The apparatus in claim 25 wherein the output device is a display.

38. The apparatus in claim 25 wherein the first and second servers are a distribution server and an advertising file server, respectively.

39. A method for use in rendering, through a computer, an information object in response to a first web page containing embedded code, the computer having a processor, a memory connected to the processor and storing both computer executable instructions and the first web page, the first web page having a plurality of computer readable instructions representing page content and the embedded code, and an output device operative in conjunction with the processor, wherein the method comprises the steps, performed by the processor and in response to the executable instructions and as a result of executing the code through a web browser, of:

downloading an agent, from a first server, into the memory and subsequently executing the agent under control of the browser, wherein the method further comprises the steps, performed by the agent, of:

downloading, from a second server and while the computer renders the first web page to a user through the output device, at least one file which is to be subsequently employed, by the processor, to render an information object;

monitoring a click-stream produced by the user to detect a user navigation event signifying a user action to

44

transition from the first web page to a next successive web page and which signifies a start of a next interstitial interval; and

in response to the user navigation event, suspending further downloading of files and processing the one file so as to render the information object through the output device to the user during the interval.

40. The method in claim 39 wherein the information object comprises a web advertisement, the code comprises advertising code and the one file comprises an advertisement file.

41. The method in claim 40 wherein the user navigation event comprises an affirmative action taken by the user, through the browser, to navigate from the first web page to the next successive web page, wherein the action comprises a mouse click, a key depression or a user-invoked state change in a stored history of web pages previously visited by the user.

42. The method in claim 41 wherein the advertisement file comprises an Ad Descriptor file or at least one advertising file specified in the Ad Descriptor file, the advertising file being either a media file or a player file.

43. The method in claim 42 further comprising the step, performed by the processor in response to executing the agent, of over-riding default life cycle methods defined in the browser with corresponding substitute methods such that the agent persistently remains in browser storage as the browser transitions across successive web pages and different web sites.

44. The method in claim 43 wherein the life cycle methods comprise at least one of start, run, stop, initialize and destroy methods.

45. The method in claim 44 wherein the agent comprises a Transition Sensor applet and an Ad Controller applet, and the method comprises the steps, performed by the processor during execution of the Transition Sensor, of:

instantiating and starting execution of the Ad Controller applet; and

monitoring the click-stream so as to detect the user-initiated event, the monitoring step comprising the steps of:

instructing the Ad Controller applet to download the Ad Descriptor file for the web advertisement from the second server into the browser storage on the computer; and

in response to an occurrence of the event, instructing the Ad Controller applet to cease any download of a further advertisement file specified in the Ad Descriptor file, to the extent any downloading of said further advertisement file is then occurring, and initiating processing through the browser, of files for an advertisement that has been previously downloaded and is currently ready to be rendered so as to render the previously downloaded advertisement during the next interstitial interval to the user.

46. The method in claim 45 further comprising the steps, performed in response to executing the advertising code, of: determining, through the agent, whether a new version of either the Transition Sensor applet or the Ad Controller applet then resides on the distribution server relative to a corresponding version, if any, of the Transition Sensor and Ad Controller applets, respectively, then residing in the browser storage; and

if said new version exists on the distribution server, downloading the new version from the distribution server into the browser storage and executing the new version in lieu of the corresponding version.

US 6,317,761 B1

45

47. The method in claim 46 wherein the Ad Controller applet comprises a play queue, wherein the method comprises the steps performed by the processor during execution of the Ad Controller applet, of:

once all the advertising files specified in an associated Ad Descriptor file for a corresponding advertisement reside in the browser storage on the computer, inserting the associated Ad Descriptor file into an end of the play queue; and

in response to the user navigation event and during the ensuing interstitial interval, processing advertising files specified in a specific Ad Descriptor file then situated at a head of the play queue so as to render, through the output device, an advertisement, corresponding to the specific Ad Descriptor file, to the user.

48. The method in claim 47 further comprising the step, performed by the agent, in response to the occurrence of the user navigation event, of generating a stop event which, when processed by the agent, suspends the downloading of further advertisement files and initiates processing of files specified in the Ad Descriptor file, then situated at the head of the play queue, so as to render the web advertisement associated therewith during the ensuing interstitial interval.

49. The method in claim 48 further comprising the step, performed by the Transition Sensor applet, of monitoring user click stream so as to detect the user navigation event and, in response thereto, produce the stop event, and processing, by the Ad Controller applet, the stop event to suspend said downloading of further advertisement files and to render the advertisement associated with the Ad Descriptor file then situated at the head of the play queue.

50. The method in claim 49 wherein the first and second servers are a distribution server and an advertising file server, respectively.

51. The method in claim 42 wherein the advertising code further comprises a component specifying the advertising server.

52. The method in claim 51 comprising the step performed by the processor, during execution of the Ad Controller applet and in response to the component contained in the code, of downloading the Ad Descriptor file originating from the advertising server specified in the second component.

53. The method in claim 42 wherein the Ad Descriptor file comprises a manifest of names of a plurality of predefined advertising files and associated configuration information necessary to properly play the downloaded advertisement through the browser.

54. The method in claim 53 wherein the Ad Descriptor file comprises a list having: a name of each player and media file that constitutes the downloaded advertisement, a corresponding network address at which said each file can be accessed, configuration information for at least one of the player files for properly configuring the corresponding player to render an associated media file.

55. The method in claim 47 further comprising the step, performed, by the processor, of processing, in response to the stored instructions and if the Ad Controller applet is not executing at the occurrence of the user navigation event, the one advertisement file, so as to render the web advertisement, only after both the Ad Controller applet has started execution and a next successive user navigation event has occurred.

56. The method in claim 47 wherein the first and second servers are a distribution server and an advertising file server, respectively.

57. The method in claim 41, wherein the advertising code comprises an advertising tag, further comprising the steps, performed by the processor, in response to execution of the tag, of:

46

dynamically writing a plurality of predefined applet tags that collectively implement a script into the first web page; and

downloading, in response to subsequent execution of the script, the agent from the first server into the memory and thereafter instantiates and executes the agent.

58. The method in claim 57 wherein the advertisement file comprises an Ad Descriptor file or at least one advertising file specified in the Ad Descriptor file, the advertising file being either a media file or a player file.

59. The method in claim 58 further comprising the step, performed by the processor, in response to the executing applet, of over-riding default life cycle methods defined in the browser with corresponding substitute methods such that the agent persistently remains in browser storage as the browser transitions across successive web pages and different web sites.

60. The method in claim 59 wherein the life cycle methods comprise at least one of start, run, stop, initialize and destroy methods.

61. The method in claim 60 wherein the agent comprises a Transition Sensor applet and an Ad Controller applet, and the method comprises the steps, performed by the processor, during execution of the Transition Sensor, of:

instantiating and starting execution of the Ad Controller applet; and

monitoring the click-stream so as to detect the user-initiated event such that the processor, wherein the monitoring step comprises the steps of:

instructing the Ad Controller applet to download the Ad Descriptor file for the web advertisement from the second server into the browser storage on the computer; and

in response to an occurrence of the event, instructing the Ad Controller applet to cease any download of a further advertisement file specified in the Ad Descriptor file, to the extent any downloading of said further advertisement file is then occurring, and initiating processing through the browser, of files for an advertisement that has been previously downloaded and is currently ready to be rendered so as to render the previously downloaded advertisement during the next interstitial interval to the user.

62. The method in claim 61 further comprising the steps, performed in the processor, in response to executing the tag, of:

determining, through the agent, whether a new version of either the Transition Sensor applet or the Ad Controller applet then resides on the distribution server relative to a corresponding version, if any, of the Transition Sensor and Ad Controller applets, respectively, then residing in the browser storage; and

if said new version exists on the distribution server, downloading the new version from the distribution server into the browser storage and executes the new version in lieu of the corresponding version.

63. The method in claim 62 wherein the Ad Controller applet comprises a play queue, wherein, the method comprises the steps, performed by the processor during execution of the Ad Controller applet, of:

once all the advertising files specified in an associated Ad Descriptor file for a corresponding advertisement reside in the browser storage on the computer, inserting the associated Ad Descriptor file into an end of the play queue; and

in response to the user navigation event and during the ensuing interstitial interval, processing advertising files



47

specified in a specific Ad Descriptor file then situated at a head of the play queue so as to render, through the output device, an advertisement, corresponding to the specific Ad Descriptor file, to the user.

64. The method in claim 63 further comprising the step, performed by the agent, in response to the occurrence of the user navigation event, of generating a stop event which, when processed by the agent, suspends the downloading of further advertisement files and initiates processing of files specified in the Ad Descriptor file, then situated at the head of the play queue, so as to render the web advertisement associated therewith during the ensuing interstitial interval.

65. The method in claim 64 further comprising the step, performed by the Transition Sensor applet, of monitoring user click stream so as to detect the user navigation event and, in response thereto, produce the stop event, and the step of processing, by the Ad Controller applet, the stop event to suspend said downloading of further advertisement files and to render the advertisement associated with the Ad Descriptor file then situated at the head of the play queue.

66. The method in claim 65 wherein the first and second servers are a distribution server and an advertising file server, respectively.

67. The method in claim 57 wherein the advertising tag further comprises first and second components, the first and second components specifying the script and the advertising server, respectively.

68. The method in claim 67 comprising the step performed by the processor, during execution of the Ad Con-

48

troller applet and in response to the second component contained in the tag, of downloading the Ad Descriptor file originating from the advertising server specified in the second component.

69. The method in claim 58 wherein the Ad Descriptor file comprises a manifest of names of a plurality of predefined advertising files and associated configuration information necessary to properly play the downloaded advertisement through the browser.

70. The method in claim 69 wherein the Ad Descriptor file comprises a list having: a name of each player and media file that constitutes the downloaded advertisement, a corresponding network address at which said each file can be accessed, configuration information for at least one of the player files for properly configuring the corresponding player to render an associated media file.

71. The method in claim 61 further comprising the step performed, by the processor, of processing, in response to the stored instructions and if the Ad Controller applet is not executing at the occurrence of the user navigation event, the one advertisement file, so as to render the web advertisement, only after both the Ad Controller applet has started execution and a next successive user navigation event has occurred.

72. The method in claim 61 wherein the first and second servers are a distribution server and an advertising file server, respectively.

\* \* \* \* \*



US006327609B1

(12) **United States Patent**  
**Ludewig et al.**

(10) **Patent No.: US 6,327,609 B1**  
(45) **Date of Patent: Dec. 4, 2001**

(54) **SYSTEM AND METHOD FOR USING COOKIES IN JAVA**

(75) Inventors: **Carl Ludewig**, Mill Valley; **Rhys Ryan**, San Francisco, both of CA (US)

(73) Assignee: **AudioBase, Inc.**, Sausalito, CA (US)

(\*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/405,446**

(22) Filed: **Sep. 22, 1999**

(51) Int. Cl.<sup>7</sup> ..... **G06F 15/16**

(52) U.S. Cl. .... **709/203; 709/223; 709/224**

(58) Field of Search ..... **709/223, 224, 709/225, 226, 229, 203, 201**

(56) **References Cited**

#### U.S. PATENT DOCUMENTS

5,774,670	*	6/1998	Montulli	709/203
5,796,952	*	8/1998	Davis et al.	709/224
5,948,061	*	9/1999	Merriman et al.	709/219
5,999,971	*	12/1999	Buckland	709/218
6,006,260	*	12/1999	Barrick, Jr. et al.	709/224
6,012,052	*	1/2000	Altschuler et al.	707/2

6,085,224	*	7/2000	Wagner	709/203
6,101,534	*	8/2000	Rothschild	709/229
6,112,240	*	8/2000	Pogue et al.	709/224
6,141,010	*	10/2000	Hoyle	345/356
6,144,944	*	11/2000	Kurtzman, II et al.	705/14
6,144,988	*	11/2000	Kappel	709/202
6,161,139	*	12/2000	Win et al.	709/225
6,163,772	*	12/2000	Kramer et al.	705/79

#### OTHER PUBLICATIONS

Joe Burns, "So You Want A Cookie, Huh?" <http://htmlgoodies.earthweb.com/tutors/cookie.html>, Jun. 18, 1997.\*

\* cited by examiner

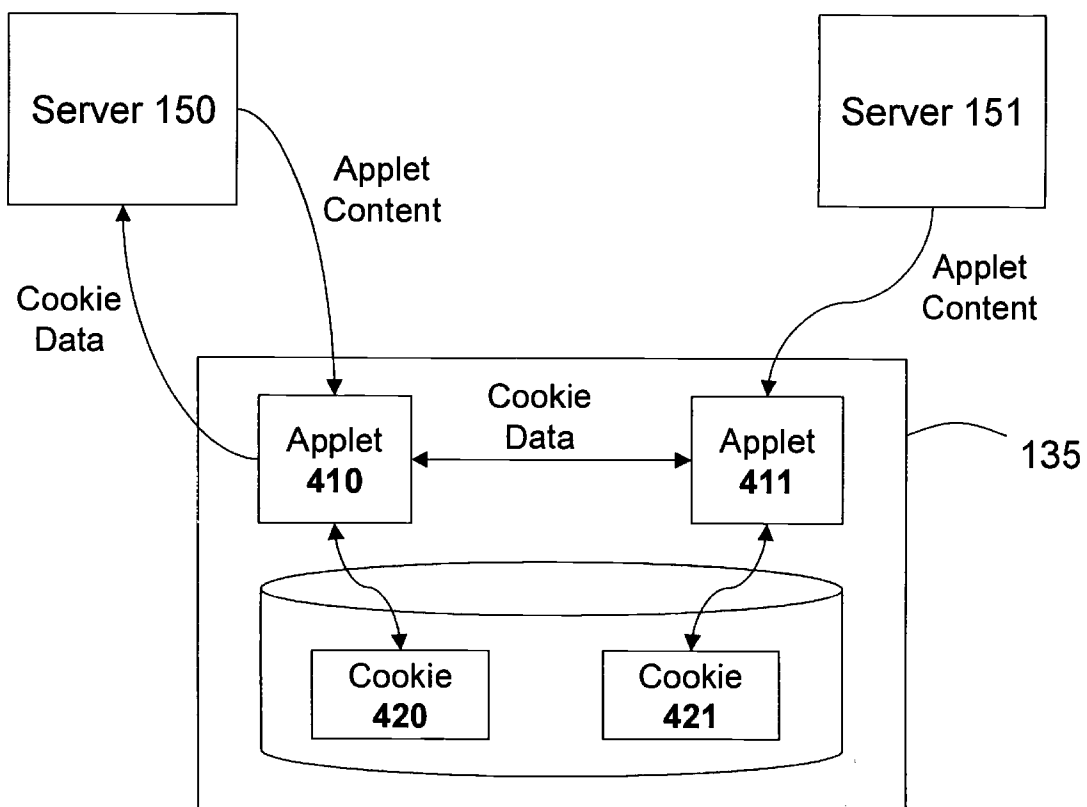
*Primary Examiner*—Mehmet B. Geckil

(74) *Attorney, Agent, or Firm*—Blakely, Sokoloff, Taylor & Zafman LLP

#### (57) **ABSTRACT**

A system for using cookies in Java is disclosed comprising a first server which: transmits an applet to a client, the applet including content of a particular type and/or subject matter and configured to store cookie data relating to the client; and subsequently interprets the cookie data to select the type and/or subject matter of applet content to transmit to the client the next time the first server transmits applet content to the client.

**16 Claims, 7 Drawing Sheets**



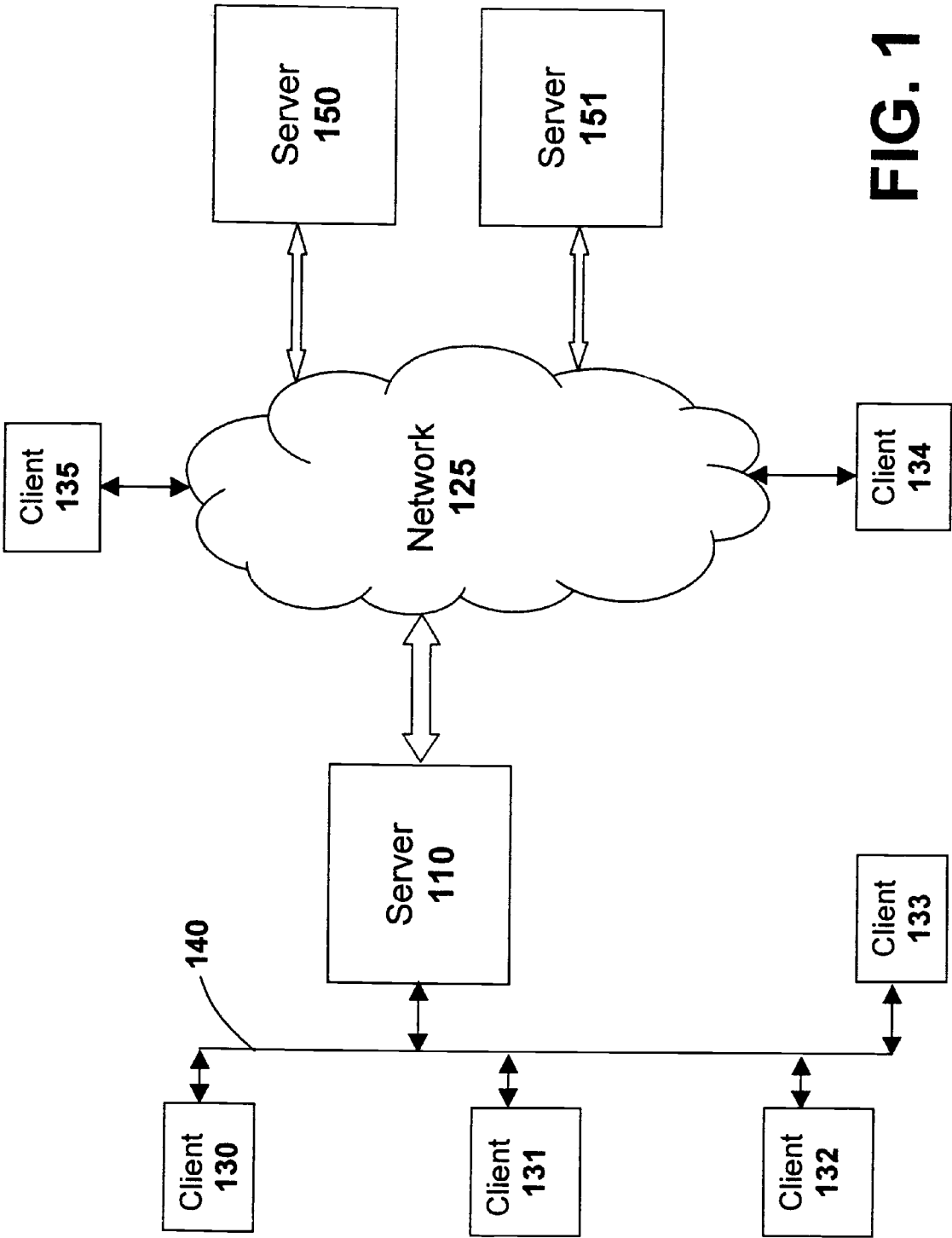


FIG. 1

200

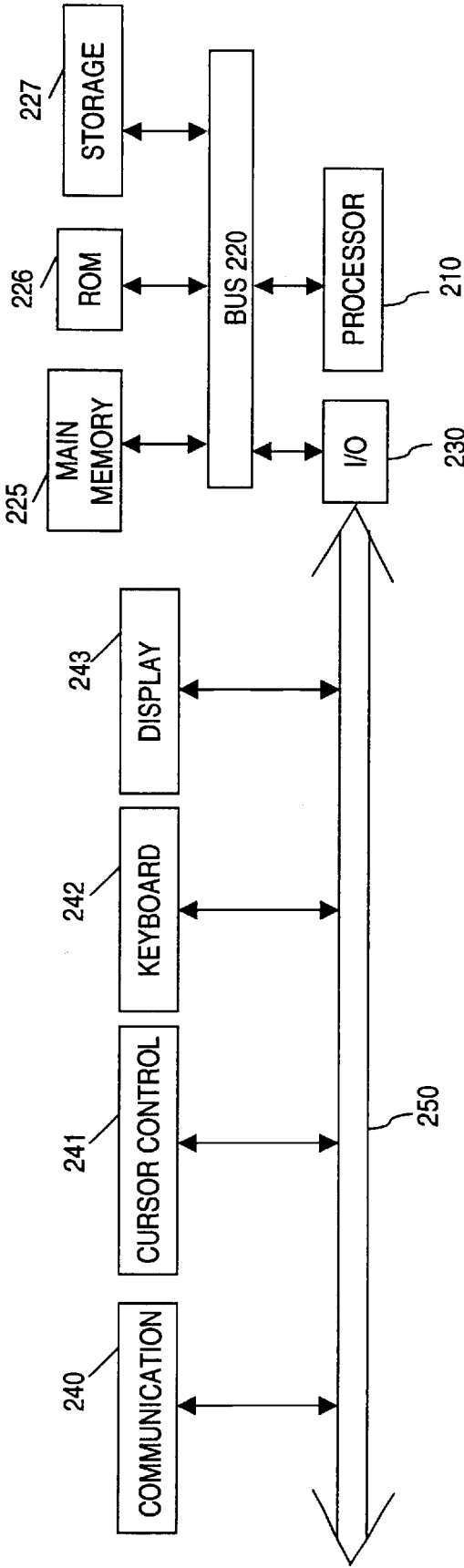


FIG. 2

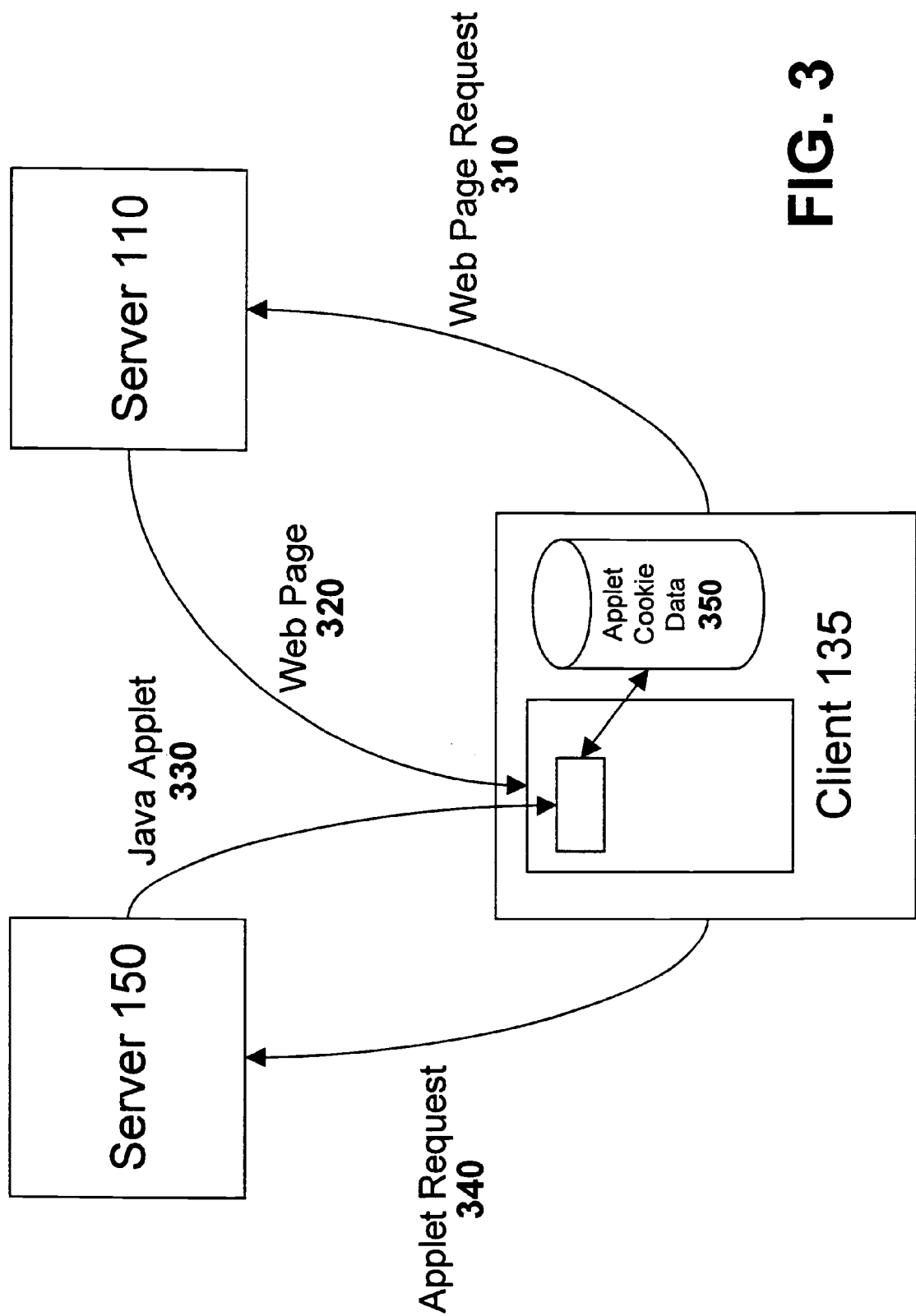


FIG. 3

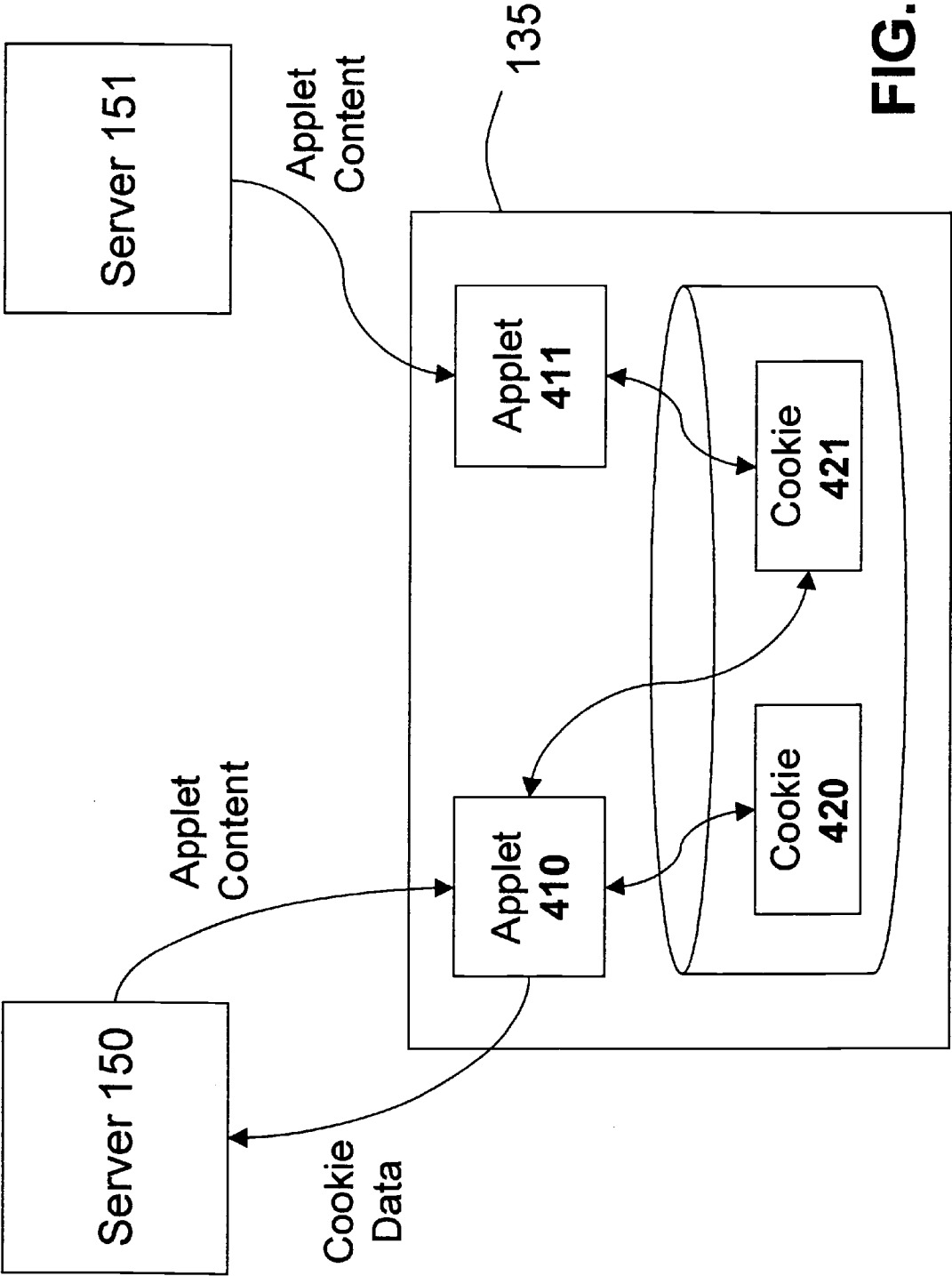


FIG. 4

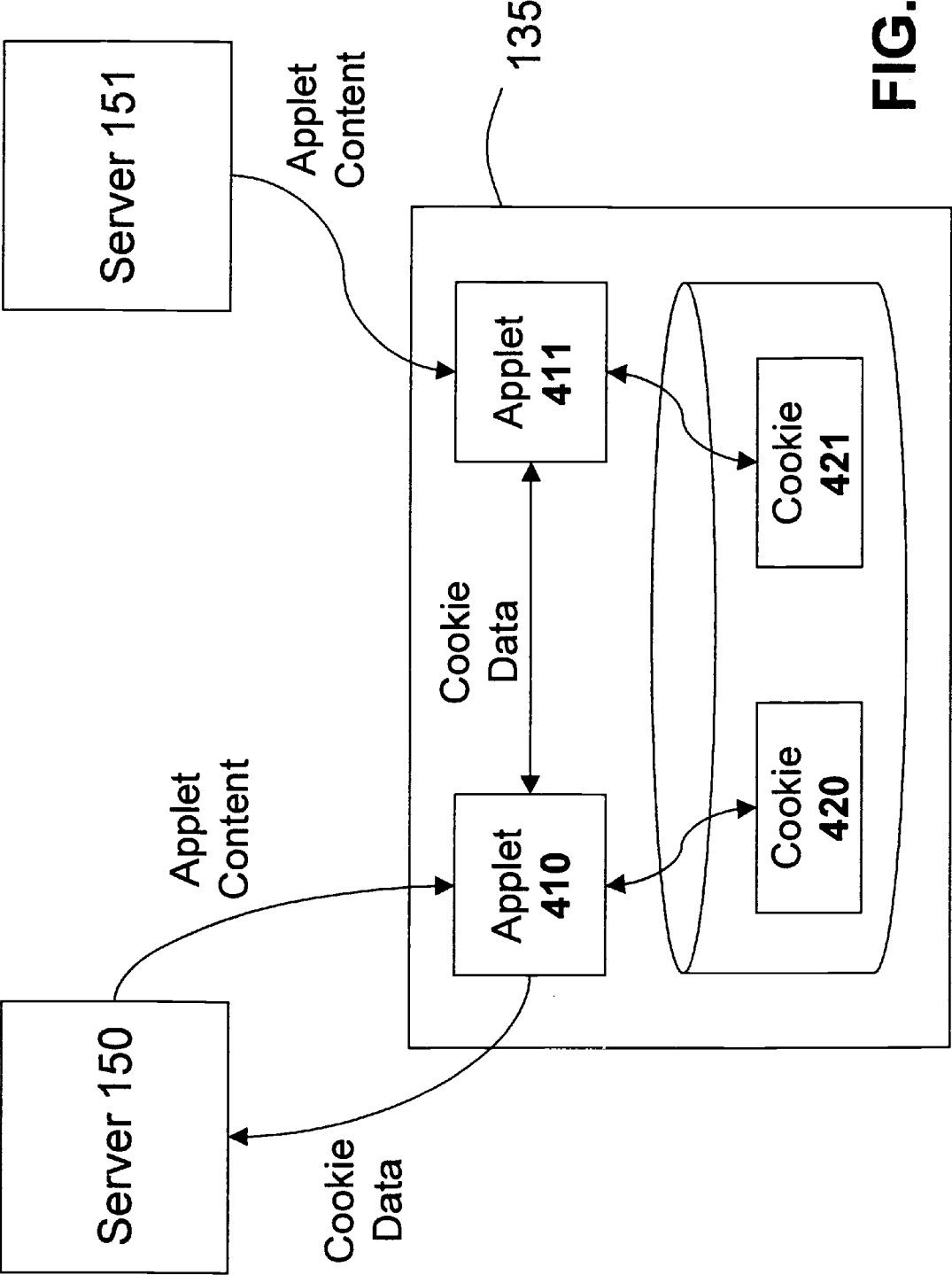


FIG. 5

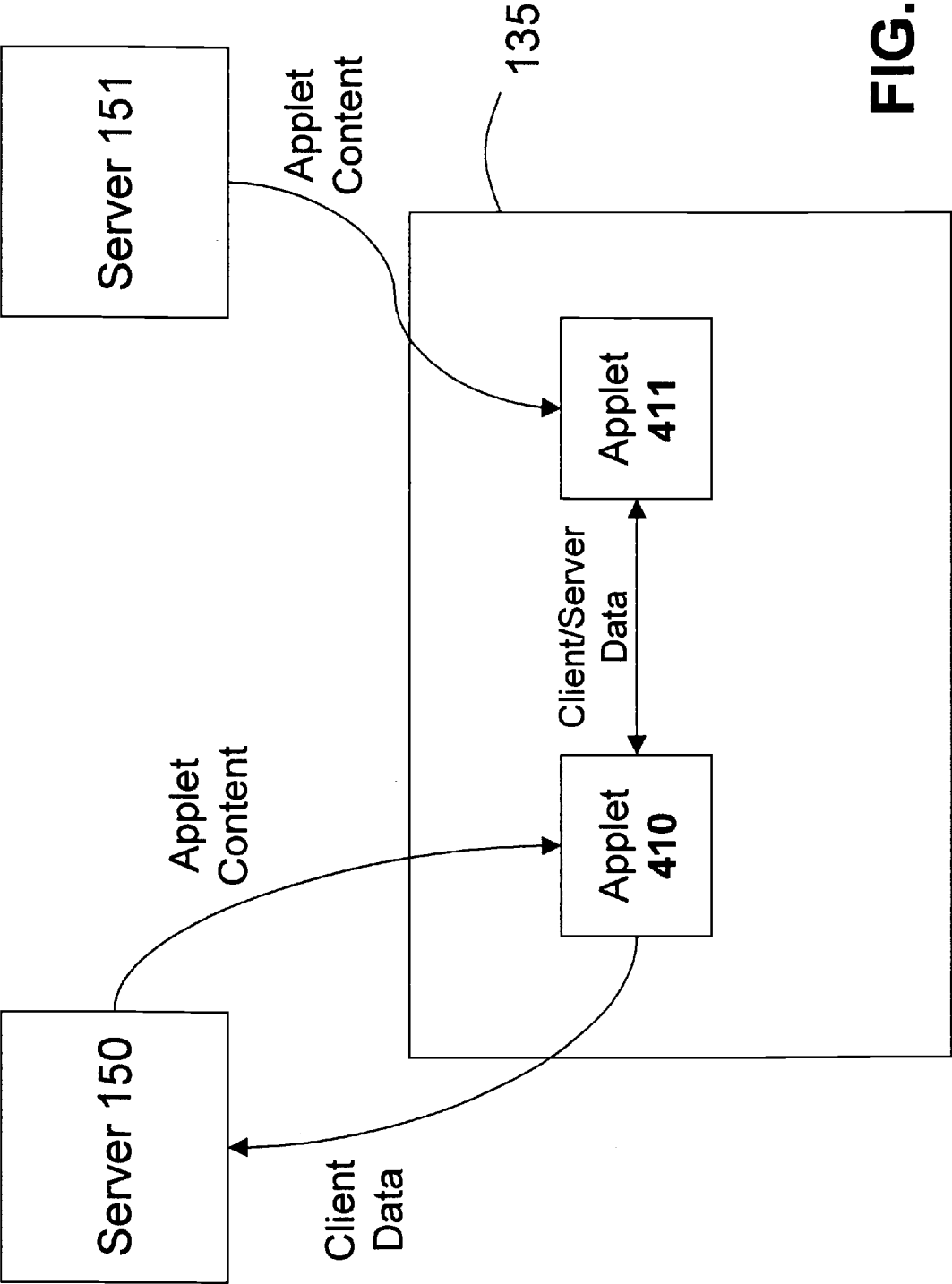
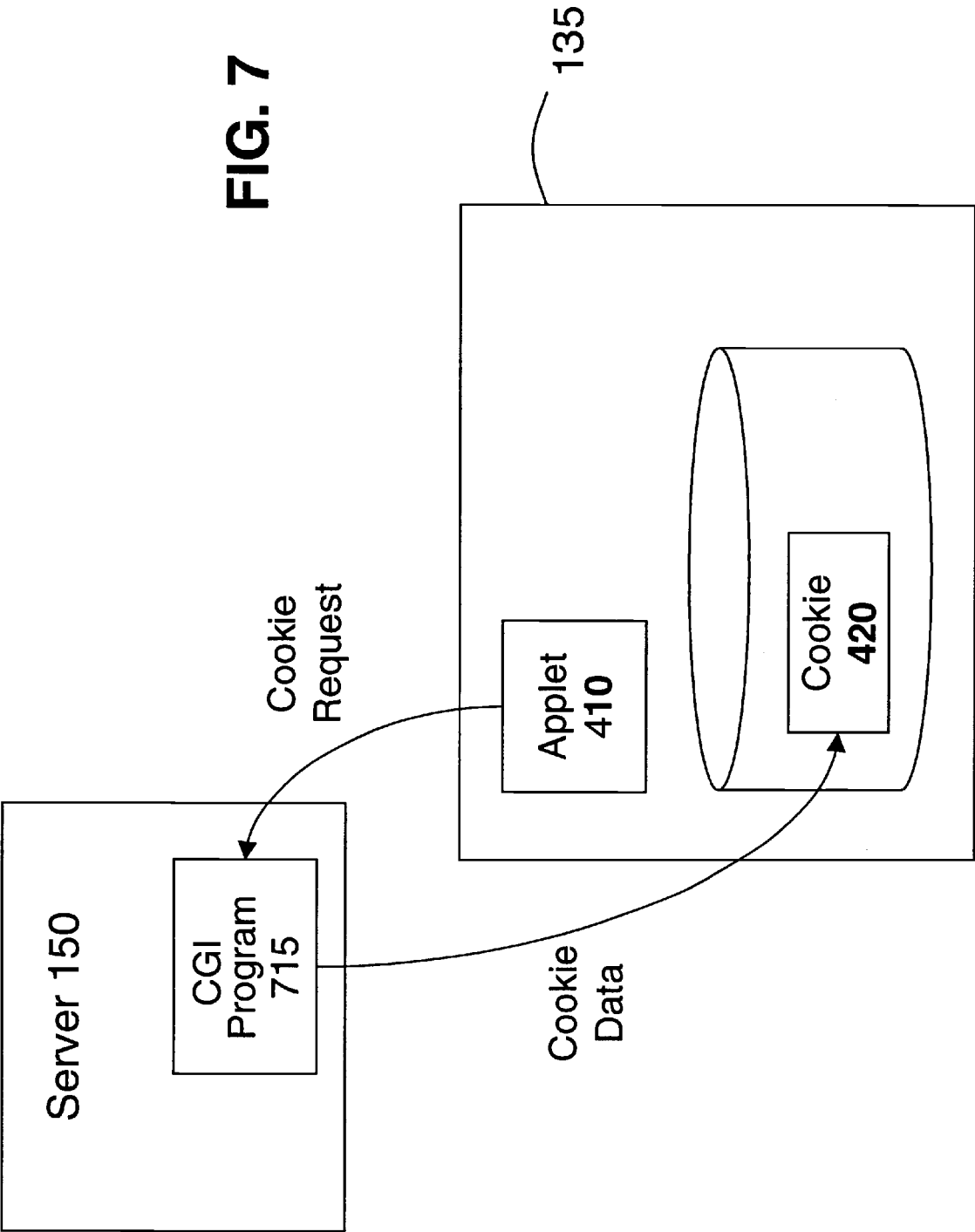


FIG. 6





US 6,327,609 B1

1

**SYSTEM AND METHOD FOR USING  
COOKIES IN JAVA**

**BACKGROUND OF THE INVENTION**

**1. Field of the Invention**

This invention relates generally to the field of Java programming. More particularly, the invention relates to an improved system and method for storing client-side data in Java.

**2. Description of the Related Art**

**Java**

Java is an object-oriented programming language which has attracted considerable amount in recent years. One key feature of Java is that it can be used to create computer programs which are platform-independent. That is, the same Java object code may be used on numerous different operating systems including Windows 95, Unix, Solaris, and Macintosh. This interoperability makes Java an ideal choice for programming Internet applications.

Once a program is written in Java source code, the Java compiler generates a compact, architecture-neutral object code known as a "bytecode." Bytecodes are executed by a runtime interpreter residing on the client computer. This runtime interpreter is commonly referred to as a Java "virtual machine." The Java virtual machine reads each bytecode and interprets the program instructions which comprise that bytecode so that the instructions may be executed by the native client microprocessor. Virtual machines are included in commonly available Internet browser applications such as Netscape Navigator™ or Microsoft Internet Explorer.™

Java was derived from the popular C++ programming language and, as such, retained many significant features of that language. For example, Java, like C++, is object-oriented. Accordingly, Java programs are developed around "classes" and "objects." These two terms are not interchangeable but they are directly related to one another. A class can be thought of as a template or blueprint from which an object is made. When an object is created from a class, new object is called a new "instance" of that class. The object will initially have all of the same characteristics of the class from which it was derived. These characteristics are defined by the data within the object and the functions and procedures—i.e., the methods—associated with the object.

"Applets" are compact Java programs comprised of one or more bytecodes. They are typically embedded in-line as objects within Hypertext Markup Language (hereinafter "HTML") documents on the World Wide Web (hereinafter "the Web") in a similar fashion to HTML-embedded images. Unlike images, however, applets may be interactive, receiving user input, executing calculations based on it and presenting continually changing content. For example, applets are ideal for generating rich media content such as audio and/or video banners. One limitation on applets, however, is that (for security reasons) they are only capable of communicating with a single Web server.

**Cookies**

Cookies are a useful tool for maintaining state information on the Internet, particularly with respect to the Web. Because the Hyper Text Transport Protocol ("HTTP")—the protocol used for communicating over the Web—is a "stateless" (i.e., non-persistent) protocol, Web servers are generally incapable of differentiating between Web site visits by clients.

2

This represents a significant limitation to client/server transactions on the Internet. In various situations, Web site owners may need their Web servers to save session information relating to Web site visits by clients. For example, e-commerce Web sites may need to assign client customers transaction IDs and passwords to help associate HTTP transactions with a single client customer. Such sites require some mechanism for determining when, and for how long, clients are "logged on." Similarly, Web sites advertisers may need to track the areas on a particular Web site most frequently visited by client customers to (for example) help build demographic databases and/or to personalize sites with dynamic content.

The "cookie" was developed to address these problems. A cookie is simply a small amount of data that a Web server saves and can later retrieve from a client system. Cookies are typically utilized by server-side code such as Common Gateway Interface ("CGI") programs. They are managed on the client side by Web browsers (referred to generally as "HTTP User Agents").

Cookies are transmitted via two HTTP headers: Cookie and Set-Cookie. Before transmitting a cookie to a client, a server typically asks the client if it is accepting cookies. If so, then the server sends the cookie data via the Set-Cookie header. Included in the header is a description of the range of Universal Resource Locators ("URLs") for which that cookie is valid. If the user subsequently revisits the Web server or any URL within the specified range, the client's browser returns the required state data via the Cookie header.

Although cookies have been widely implemented for Web pages, they have not been used to store state information associated with Java applets. Due to the widespread use of applets embedded in HTML documents on the Internet, and due to the fact that these applets are frequently uploaded to the client by a server other than the server on which the HTML document resides, a system for using cookies with Java applets would be beneficial to the applet provider for at least the reasons stated above. For example, such a system would allow a Web server providing advertising applets to keep track of the different applets (e.g., multimedia ads, sound-bytes . . . etc) downloaded by a particular client.

In addition, it would be useful in some circumstances to configure an applet (e.g., downloaded from a first Web server) to read cookie data generated by a other applets (e.g., downloaded from other Web servers). The first Web server could then learn more about the particular client and could narrowly tailor applet data for the client (e.g., sound or video clips) which accurately matches the client's preferences.

Accordingly, what is needed is a system and method for using cookies in Java. What is also needed is a system and method for using cookies in Java wherein two applets generated by different Web server can exchange cookie data. What is also needed is a system and method for using cookies in Java which can be implemented on currently existing Java platforms.

**SUMMARY OF THE INVENTION**

A system for using cookies in Java is disclosed comprising a first server which: transmits an applet to a client, the applet including content of a particular type and/or subject matter and configured to store cookie data relating to the client; and subsequently interprets the cookie data to select the type and/or subject matter of applet content to transmit to the client the next time the first server transmits applet content to the client.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention can be obtained from the following detailed description in conjunction with the following drawings, in which:

FIG. 1 illustrates an exemplary network architecture used to implement elements of the invention.

FIG. 2 illustrates an exemplary computer architecture used to implement elements of the invention.

FIG. 3 illustrates one embodiment of a system for using cookies in Java.

FIG. 4 illustrates another embodiment of a system for using cookies in Java.

FIG. 5 illustrates another embodiment of a system for using cookies in Java.

FIG. 6 illustrates an embodiment in which two applets exchange information directly.

FIG. 7 illustrates an embodiment in which an applet causes cookie data to be stored indirectly.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

A System and Method for Using Cookies in Java

An improved system and method is described for streaming Java. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the invention. It will be apparent, however, to one skilled in the art that the invention may be practiced without some of these specific details. In other instances, well-known structures and devices are shown in block diagram form to avoid obscuring the underlying principles of the invention.

Embodiments of the invention include various steps, which will be described below. The steps may be embodied in machine-executable instructions. The instructions can be used to cause a general-purpose or special-purpose processor which is programmed with the instructions to perform certain steps. Alternatively, these steps may be performed by specific hardware components that contain hardwired logic for performing the steps, or by any combination of programmed computer components and custom hardware components.

Elements of the present invention may be provided as a computer program product which may include a machine-readable medium having stored thereon instructions which may be used to program a computer (or other electronic device) to perform a process. The machine-readable medium may include, but is not limited to, floppy diskettes, optical disks, CD-ROMs, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, magnet or optical cards, propagation media or other type of media/machine-readable medium suitable for storing electronic instructions. For example, the present invention may be downloaded as a computer program product, wherein the program may be transferred from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way of data signals embodied in a carrier wave or other propagation medium via a communication link (e.g., a modem or network connection).

An Exemplary Network Architecture

Elements of the present invention may be included within a client-server based system **200** such as that illustrated in FIG. 1. According to the embodiment depicted in FIG. 1, one

or more servers **110, 150**, and **151** communicate to one or more clients **130–135**. The clients **130–135** may transmit and receive data from the servers **110, 150**, and **151** over a variety of communication media including (but not limited to) a local area network **140** and/or a larger network **125** (e.g., the Internet). Alternative communication channels such as wireless communication via satellite broadcast (not shown) are also contemplated within the scope of the present invention.

The servers **110, 150**, and **151** may include one or more databases for storing Java code and/or digital audio and/or video data. The databases may also store specific client data (e.g., information on how frequently a particular client logs in to server **110** and that client's preferences) and/or more general data. The database in one embodiment runs an instance of a Relational Database Management System (RDBMS), such as Microsoft™ SQL-Server, Oracle™ or the like.

A client may interact with and receive feedback from servers **110, 150**, and **151** using various different communication devices and/or protocols. In one embodiment, the client logs in to servers **110, 150**, and **151** via client software. The client software may include a browser application which supports Java such as Netscape Navigator™ or Microsoft Internet Explorer™ on the client's personal computer and may communicate to servers **110, 150, 151** via the Hypertext Transfer Protocol (hereinafter "HTTP"). In other embodiments included within the scope of the invention, clients may communicate with servers **110, 150, 151** via cellular phones and pagers (e.g., in which the necessary software is embedded in a microchip), handheld computing devices, and/or touch-tone telephones.

An Exemplary Computer Architecture

Having briefly described an exemplary network architecture which employs various elements of the present system and method, a computer system **200** representing exemplary clients **130–135** and/or servers **110, 150**, and **151** in which elements of the system and method may be implemented will now be described with reference to FIG. 2.

One embodiment of a computer system **200** comprises a system bus **220** for communicating information, and a processor **210** coupled to bus **220** for processing information. Computer system **200** further comprises a random access memory (RAM) or other dynamic storage device **225** (referred to herein as main memory), coupled to bus **220** for storing information and instructions to be executed by processor **210**. Main memory **225** also may be used for storing temporary variables or other intermediate information during execution of instructions by processor **210**. Computer system **200** also may include a read only memory (ROM) and/or other static storage device **226** coupled to bus **220** for storing static information and instructions used by processor **210**.

A data storage device **227** such as a magnetic disk or optical disc and its corresponding drive may also be coupled to computer system **200** for storing information and instructions. Computer system **200** can also be coupled to a second I/O bus **250** via and I/O interface **230**. A plurality of I/O devices may be coupled to I/O bus **250**, including a display device **243**, an input device (e.g., an alphanumeric input device **242** and/or a cursor control device **241**).

The communication device **240** may comprise a modem, a network interface card, or other well known interface device, such as those used for coupling to Ethernet, token ring, or other types of networks. In any event, in this manner,

the computer system 200 may be coupled to a number of servers via a conventional network infrastructure, such as a company's local area network 140 and/or a larger network 125, for example.

One Embodiment of the System and Method for  
Using Cookies in Java

FIG. 3 illustrates client 135 communicating over a network 125 to servers 150 and 110. In one embodiment of the system, client 135 initially makes a Web page request 310 from server 110 and, in response, server 110 transmits the requested Web page 320 to client 135. The Web page request 310 may contain more information than a simple Web page address. For example, if server 110 is a search engine (e.g., such as Excite™ or Yahoo™), then Web page request 310 may contain a query for specific information (e.g., "mountain bikes," "airline tickets" . . . etc). In response, server 110 sends a Web page 320 to client 135 which contains links to Web sites that contain information related to the query (e.g., "www.trek bikes.com," "www.continental.com" . . . etc).

Web page 320 may also include embedded Web objects such as audio or video applets which are automatically transmitted to the client 135 when the Web page is downloaded. This is illustrated in FIG. 3 as applet request 340 from client 135 to server 150, and subsequent Java applet 330 download. In one embodiment of the system, server 150 is an advertisement server (i.e., a server which provides embedded Web page advertisements) and the particular Java applet transmitted to client 135 is selected based on the type of Web page 320 transmitted to client 135 from server 110. For example, if client 135 downloaded a Web page 320 with links relating to mountain biking (e.g., in response to a query for "mountain biking"), the Java applet 330 may include sound and/or video content for a particular brand of mountain bike. It should be noted, however, that the specific configuration illustrated in FIG. 3 is for the purpose of example only. Such as configuration (i.e., an applet embedded in a Web page) is not necessary to comply with the underlying principles of the invention.

In one embodiment of the system, Java applet 330 stores cookie data 350 relating to the preferences of client 135. For example, when client 135 downloads a Java applet stored on server 150, the applet cookie 350 on client 135 is updated to include data identifying the subject-matter of the applet (or, e.g., the Web page in which the applet is embedded). The next time client 135 requests an applet from server 150, the request will include the stored cookie data 350. As a result, server 150 will transmit an applet including subject-specific audio and/or video content based on client 135's stored preferences.

Cookie data resulting from numerous similar applet downloads may be analyzed on server 150 (e.g., based on compiled marketing research) so that future applet transmissions to client 135 are uniquely tailored to client 135's interests. For example, if a significant percentage of previous applet downloads included subject matter on outdoor activities (e.g., mountain biking, camping, hiking . . . etc) then the next time client 135 transmits a request 340 for an embedded audio and/or video applet, server 150 may transmit an audio applet for Jeep Wrangle™ (or, alternatively, for a Jeep Grand Cherokee™ if the market-based cookie analysis on server 150 indicates that client 135 is in a higher income bracket).

Another embodiment of a system for using cookies in Java is illustrated in FIG. 4. In this embodiment, two separate applets 410 and 411 store cookie data 420 and 421,

respectively. The individual cookie data 420 and 421 stored for each applet 410, 411 may be used by servers 150 and 151, respectively, in the same manner described above (e.g., to determine the subject-matter for applet content).

5 In addition, in this embodiment individual cookie data 420,421 may be exchanged between the applets 410, 411. For example, as illustrated in FIG. 4, applet 410 may read cookie data from applet 411's cookie file 421 (as well as it's own cookie file 420). One benefit achieved through this embodiment is that server 150 is provided with supplemental information about client 135's preferences, and can therefore transmit applet content which is more narrowly tailored to client 135's taste. It should also be noted that this embodiment effectively circumvents the limitation that one  
10 applet may only communicate to one Web server.

In one embodiment of the system, both server 150 and server 151 are advertisement servers that provide audio and/or video content to Web pages. In this embodiment, each of the cookie files 420,421 may contain different types of marketing data about client 135. For example (returning to the previous example) cookie 420 may contain data indicating that client 135 prefers outdoor activities. By contrast, cookie 421 may contain information on the airline that client 421 prefers to use when traveling. Accordingly, in response to the combined data, server 510 may provide an audio and/or video advertisement of reduced fares to wilderness destinations on client 421's favorite airline.

Rather than reading the cookie data 421 of applet 411 directly, in the embodiment of the system illustrated in FIG. 5, applet 410 communicates directly with applet 411 to retrieve cookie data. Regardless of which embodiment of the system is implemented, however, the underlying principles remain the same.

35 In another embodiment, illustrated in FIG. 6, the two applets 410,411 exchange non-cookie data. Thus, if client 135 has cookies disabled on his computer, this embodiment will still allow server 150 to obtain information about client 135 collected by applet 411. This may be any of the types of information discussed above except in a non-cookie format.

40 In one embodiment of the system and method, the applet does not store or manipulate cookies directly. Rather, it makes a request back to the server and the server sets a cookie in the browser. For example, as illustrated in FIG. 7, the applet 410 may request that a cookie be sent from a common gateway interface ("CGI") program 710, which sets a cookie 420 in the browser (not shown). Subsequent requests from the browser or applet 410 will then return the cookie 420 to the server. Thus, in this embodiment, the applet 410 never directly manipulates the cookie 420, but by making the appropriate requests to the server 150, the applet 410 can cause the underlying browser to set/get the cookie 420. This embodiment is particularly useful for system where applets are not permitted (e.g., for security reasons) to read or write to the file system of the client 135. If the applet 410 needed the exact value of cookie 420, it might be sent to the applet 410 by the server 150 as text data.

65 Throughout the foregoing description, for the purposes of explanation, numerous specific details were set forth in order to provide a thorough understanding of the invention. It will be apparent, however, to one skilled in the art that the invention may be practiced without some of these specific details. For example, FIG. 3 illustrates two servers and a single client for implementing specific aspects of the invention. However, one of ordinary skill in the art will recognize that any number of servers (i.e., one or more) and/or clients may implement aspects of the invention. The systems illus-

7

trated in FIGS. 3 through 5 are merely a few exemplary embodiments. In addition, while the foregoing description focused on the Java programming language, other platform-independent programming languages may be used without departing from the underlying principles of the invention. 5 Accordingly, the scope and spirit of the invention should be judged in terms of the claims which follow.

- What is claimed is:
1. A system for using cookies in Java comprising a first server which:
    - transmits an applet to a client, said applet including content of a particular type and/or subject matter;
    - receives a message from said applet instructing said first server to store cookie data on said client;
    - subsequently interprets said cookie data to select the type and/or subject matter of applet content to transmit to said client the next time said first server transmits applet content to said client; and
    - interprets cookie data stored by a second applet to select the type of applet content to transmit to said client, said second applet being transmitted to said client by a second server.
  2. The system as claimed in claim 1 wherein said applet content type is digital audio content.
  3. The system as claimed in claim 1 wherein said applet content type is digital video content.
  4. The system as claimed in claim 1 wherein said first server is an advertisement server.
  5. The system as claimed in claim 1 wherein said second server is an advertisement server.
  6. The system as claimed in claim 1 wherein said applet is embedded in an HTML document.
  7. A method for using cookies in Java comprising:
    - transmitting a first applet from a first server to a client, said first applet adapted to provide digital audio/video content to said client and to request that said first server store cookie data on said client relating to said client; said first server storing said cookie data on said client in response to said request;
    - transmitting a second applet from a second server to said client, said second applet adapted to request that said second server store cookie data on said client relating to said client;

8

- said first applet communicating directly with a second applet to retrieve said cookie data stored by said second server;
  - determining said client's preferences based on said cookie data stored by said second server and said first server;
  - said first applet instructing said first server to update said cookie data based on said client's preferences; and
  - selecting additional audio/video content to transmit to said client based on said updated cookie data.
8. The method for using cookies in Java as claimed in claim 7 wherein said first applet exchanges non-cookie data with said second applet, said non-cookie relating to said client's preferences.
  9. The method for using cookies in Java as claimed in claim 7 wherein said first server is an advertisement server.
  10. The method for using cookies in Java as claimed in claim 7 wherein said second server is an advertisement server.
  11. The system as claimed in claim 7 wherein said first applet is embedded in an HTML document.
  12. An article of manufacture including a sequence of instructions stored on a machine-readable media which when executed by a processor cause the processor to:
    - retrieve a first Java applet from a first server, said first Java applet configured to cause said first server to store cookie data relating to a first client;
    - retrieve a second Java applet from a second server, said second Java applet configured to read said cookie data stored by said first server through direct communication of said cookie data from said first Java applet to said second Java applet.
  13. The article of manufacture as in claim 12 comprising additional instructions which cause said processor to:
    - provide direct communication of non-cookie data from said first Java applet to said second Java applet.
  14. The article of manufacture as claimed in claim 12 wherein said first server is an advertisement server.
  15. The article of manufacture as claimed in claim 12 wherein said second server is an advertisement server.
  16. The article of manufacture as claimed in claim 12 wherein said first applet is embedded in an HTML document.

\* \* \* \* \*





US006401134B1

(12) **United States Patent**  
**Razavi et al.**

(10) **Patent No.:** **US 6,401,134 B1**  
(45) **Date of Patent:** **Jun. 4, 2002**

(54) **DETACHABLE JAVA APPLET**  
(75) Inventors: **Behfar Razavi**, San Jose; **Eric Harshbarger**, San Francisco, both of CA (US)  
(73) Assignee: **Sun Microsystems, Inc.**, Mountain View, CA (US)  
(\*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/910,481**  
(22) Filed: **Jul. 25, 1997**  
(51) **Int. Cl.**<sup>7</sup> ..... **G06F 9/54**  
(52) **U.S. Cl.** ..... **709/310**; 345/808  
(58) **Field of Search** ..... 345/326–357,  
345/163, 700, 764, 804, 808; 709/200,  
310–332; 707/513

(56) **References Cited**  
**U.S. PATENT DOCUMENTS**  
4,931,783 A \* 6/1990 Atkinson ..... 345/163  
5,742,768 A \* 4/1998 Gennaro et al. .... 709/200  
5,802,530 A \* 9/1998 Van Hoff ..... 707/513  
5,818,445 A \* 10/1998 Sanderson et al. .... 345/334  
5,870,091 A \* 2/1999 Lazarony et al. .... 345/346  
5,923,326 A \* 7/1999 Bittinger et al. .... 345/340

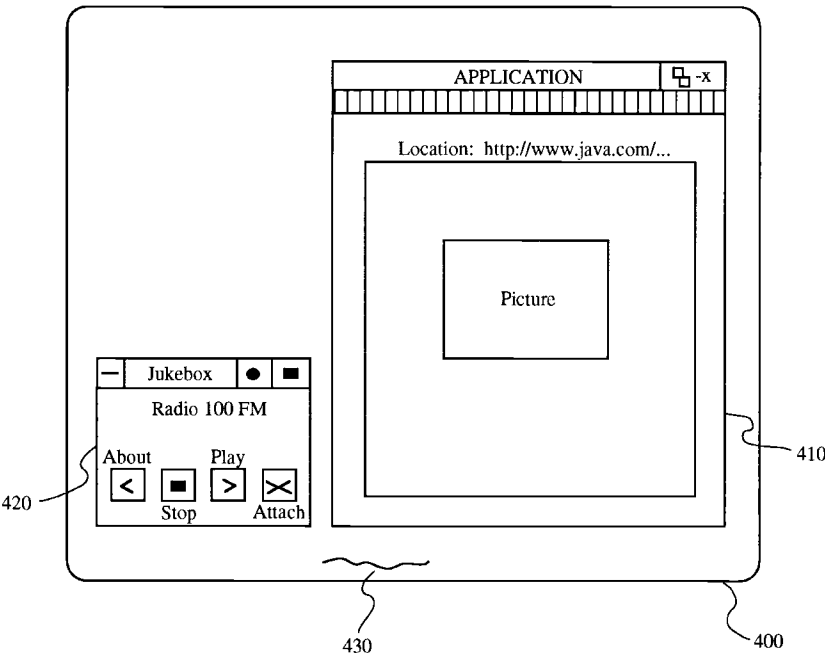
**OTHER PUBLICATIONS**  
David Mitchell, "Leveraging your Visual C++ experience on the Internet with thin client technology," Microsoft Systems Journal, v11,n12,p47(13), 12/96.\*

Lisa Nadile, "Microsoft expanding OCX: readies detachable, embeddable applets for the Internet", PC Week, v13, n8, p6(1) 2/96.\*  
Jugel et al; "The Java Telnet applet: documentation", 1996.\*  
Morrison et al, Java Unleashed, 12/96.\*  
Afergan et al., Web programming desktop reference, chap. 14, 1996.\*  
Gosling et al, Java API documentation, version 1.0.2, 1996.\*  
Jamie Jaworski, JAVA Developer Guide, chapters 1,15,16, 19,31, and appendix, 1996.\*  
Warth et al, User's guide to JDK (beta 1.0): applet; Oct. 24, 1995.\*  
author unknown, Java API documentation, Feb. 22, 1996.\*  
\* cited by examiner

*Primary Examiner*—Dung C. Dinh  
(74) *Attorney, Agent, or Firm*—Blakely Sokoloff Taylor & Zafman

(57) **ABSTRACT**  
A method and system is disclosed for detaching Java applets from the constraints of the application such as a browser which provides the Java engine for executing those applets. Thus, the applets, when detached, can appear in a detached window which is more easily controllable by the operating environment desktop. The Java applets continue to run under the application's virtual machine but do so without regard to the graphical interface limits of the application. Further, if the application that launched the applet proceeds to a new URL location, the Java applet continues to run. Also, the applet, once detached, can be reattached into the application to appear in the application history.

**20 Claims, 7 Drawing Sheets**





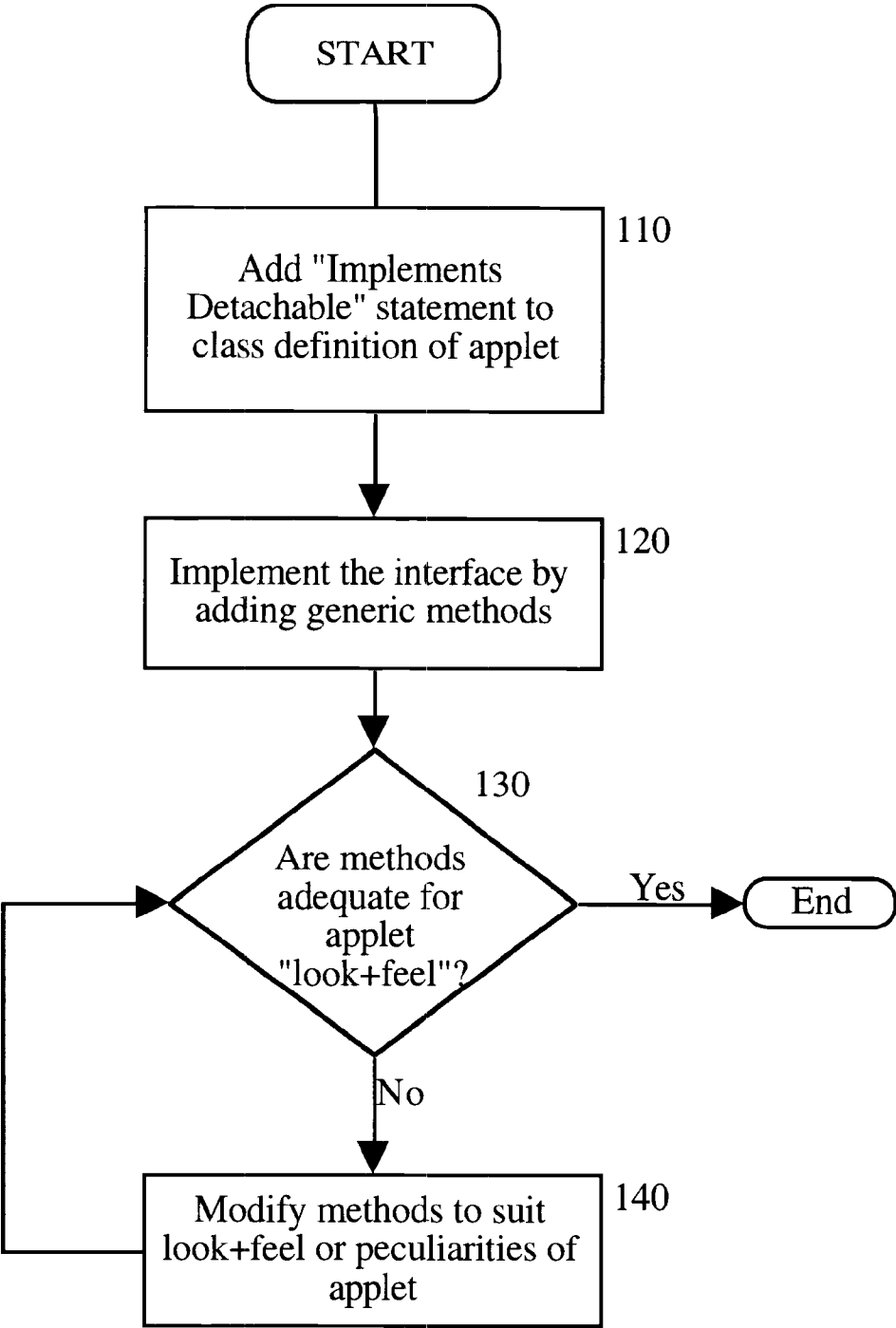


FIGURE 1

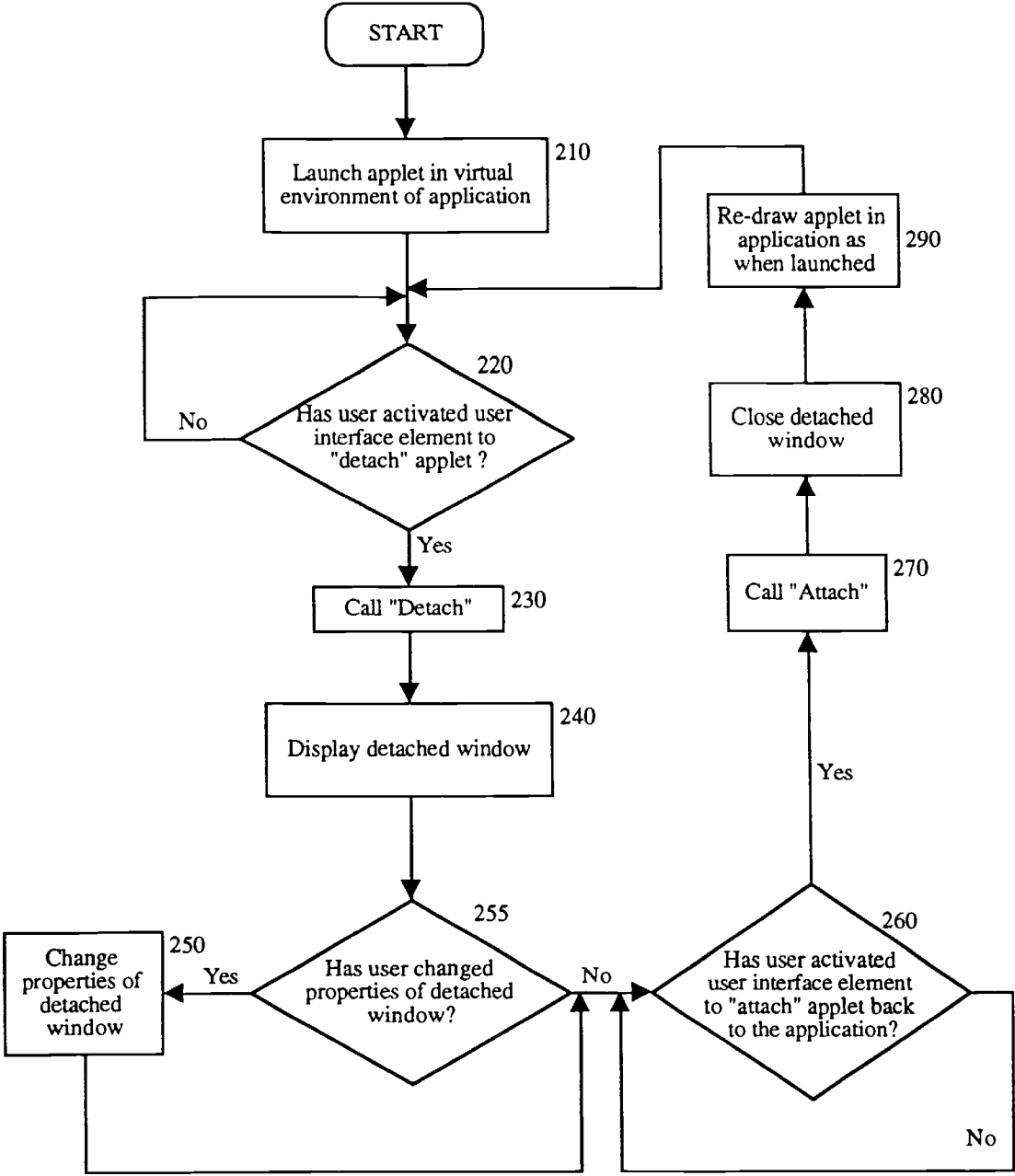


FIGURE 2

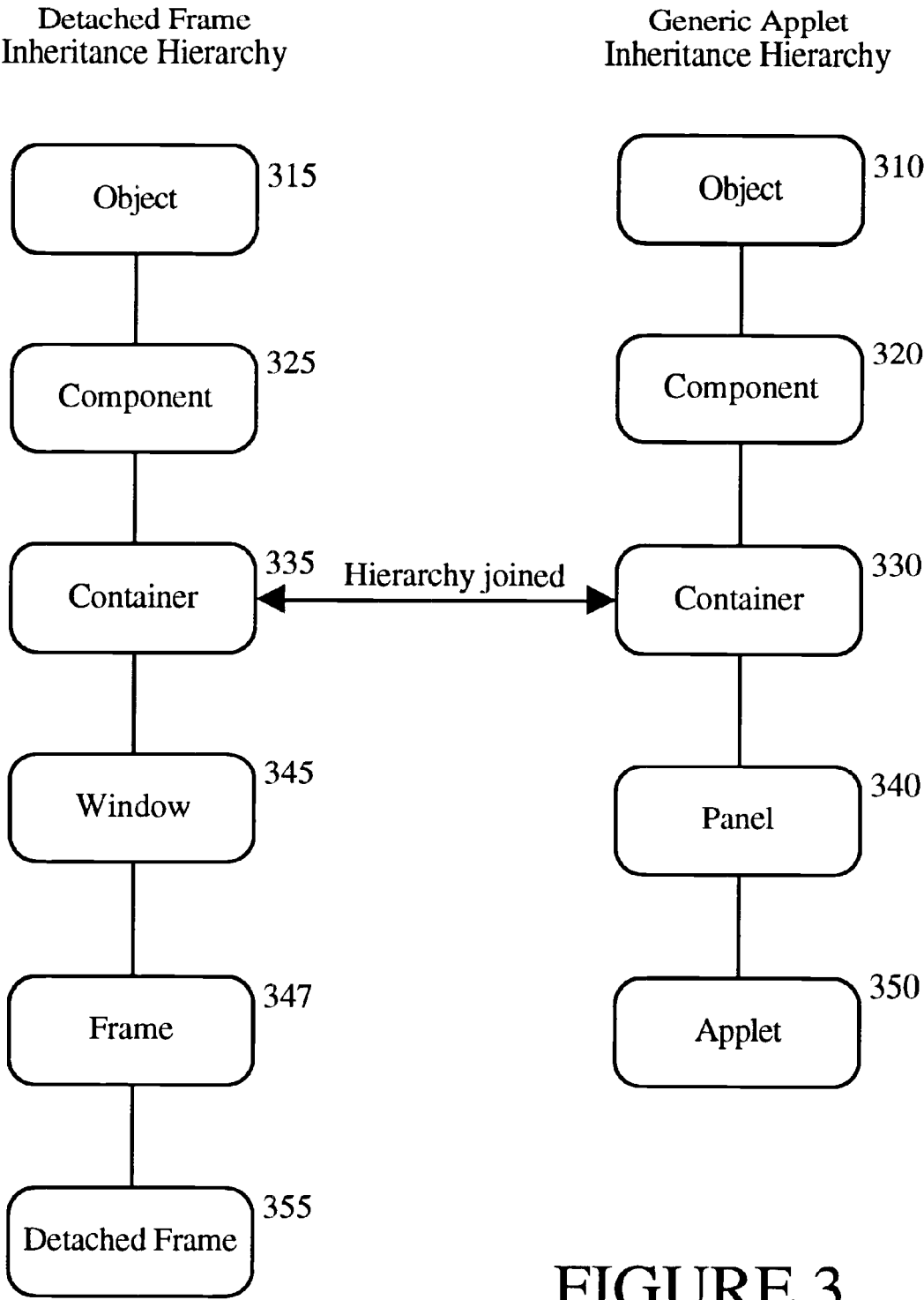


FIGURE 3

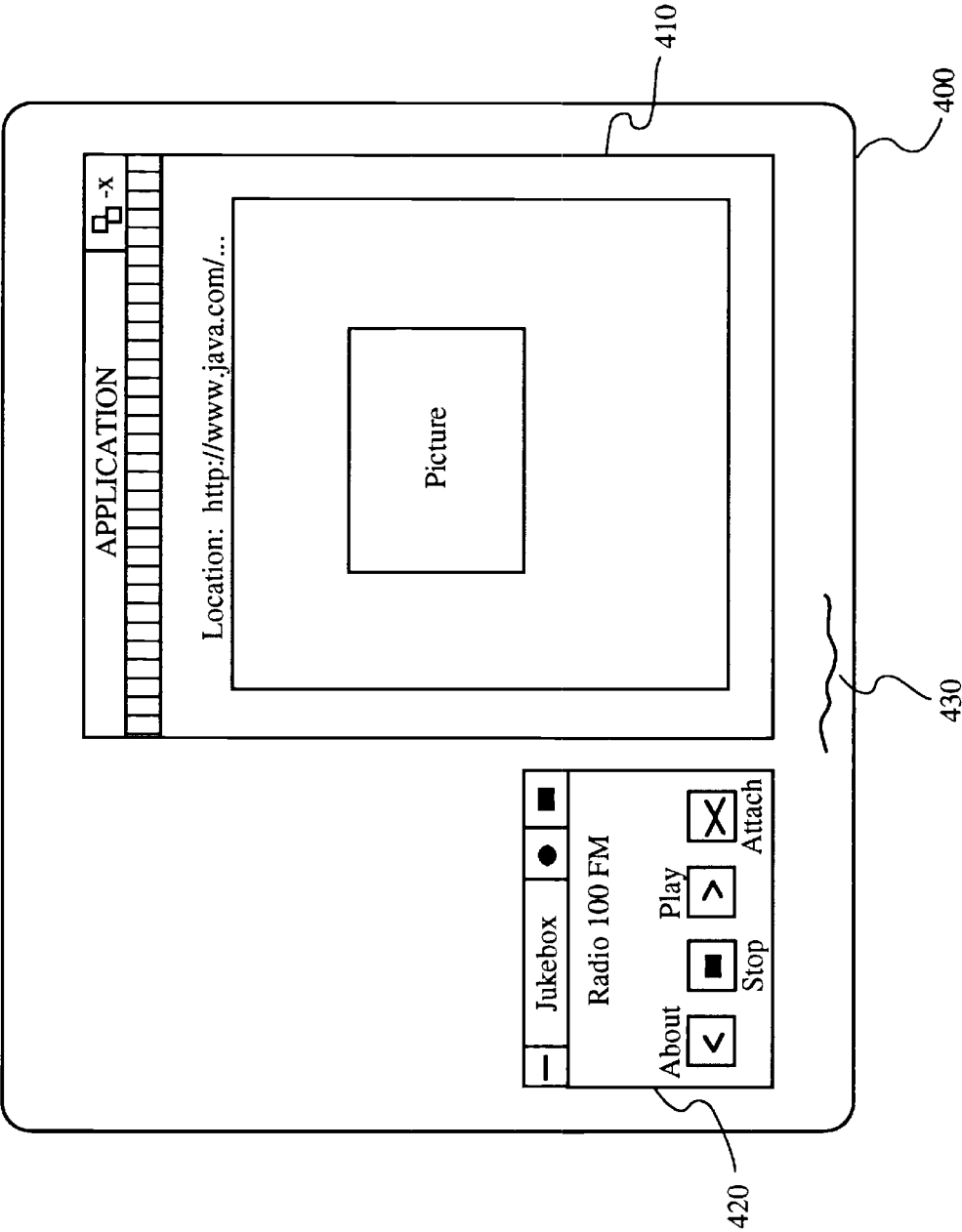


FIGURE 4

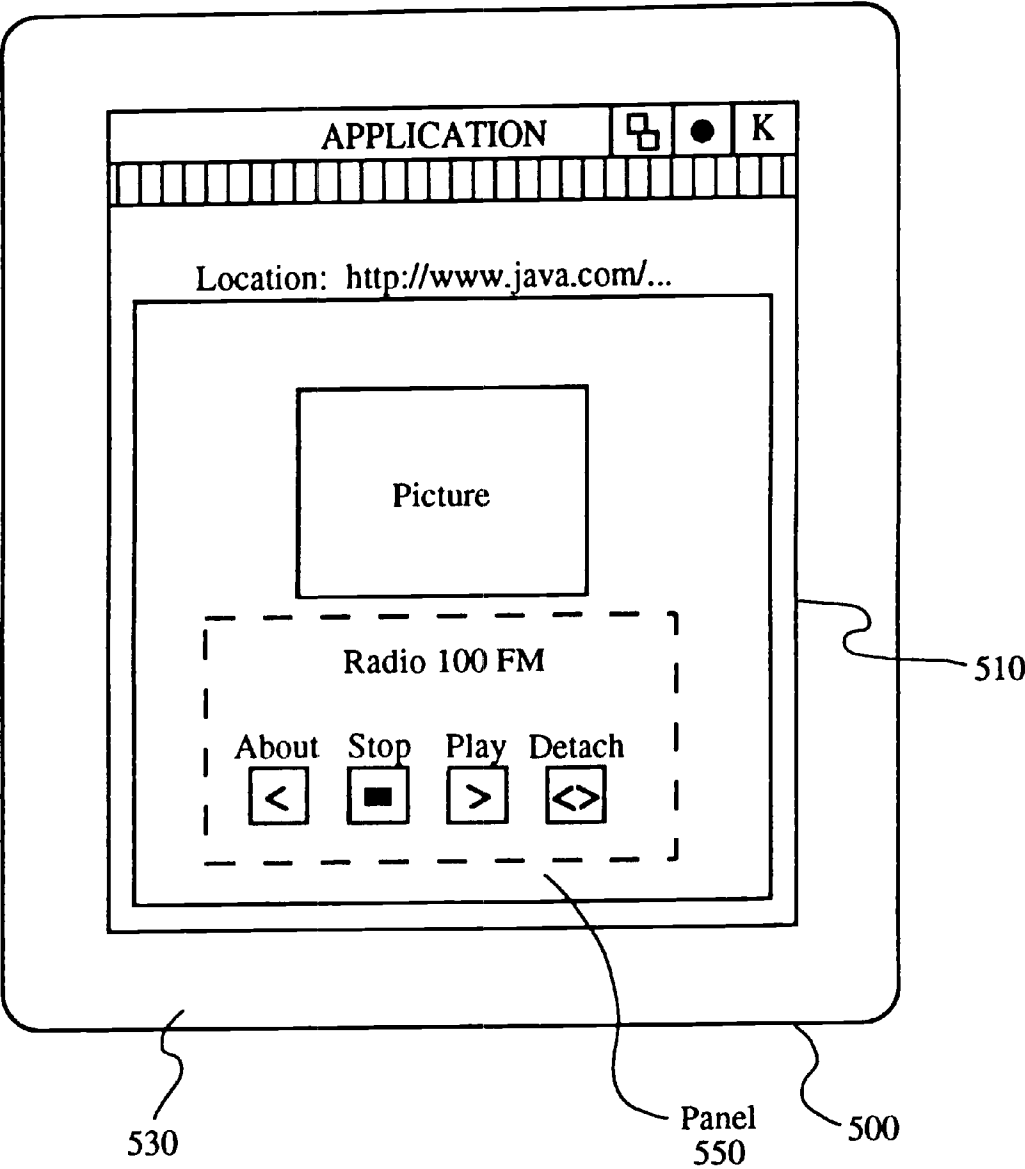


FIGURE 5

U.S. Patent

Jun. 4, 2002

Sheet 6 of 7

US 6,401,134 B1

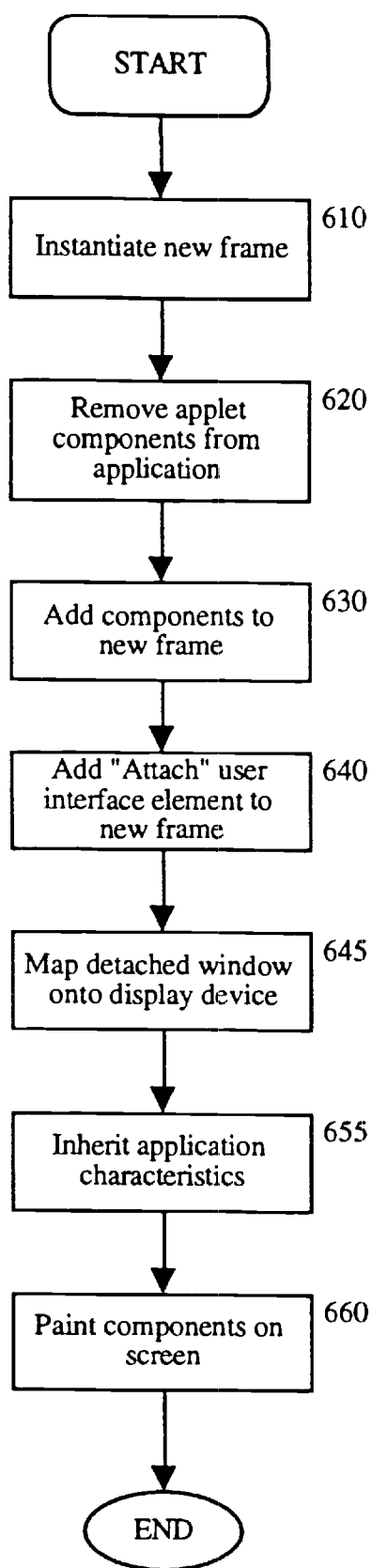


FIGURE 6



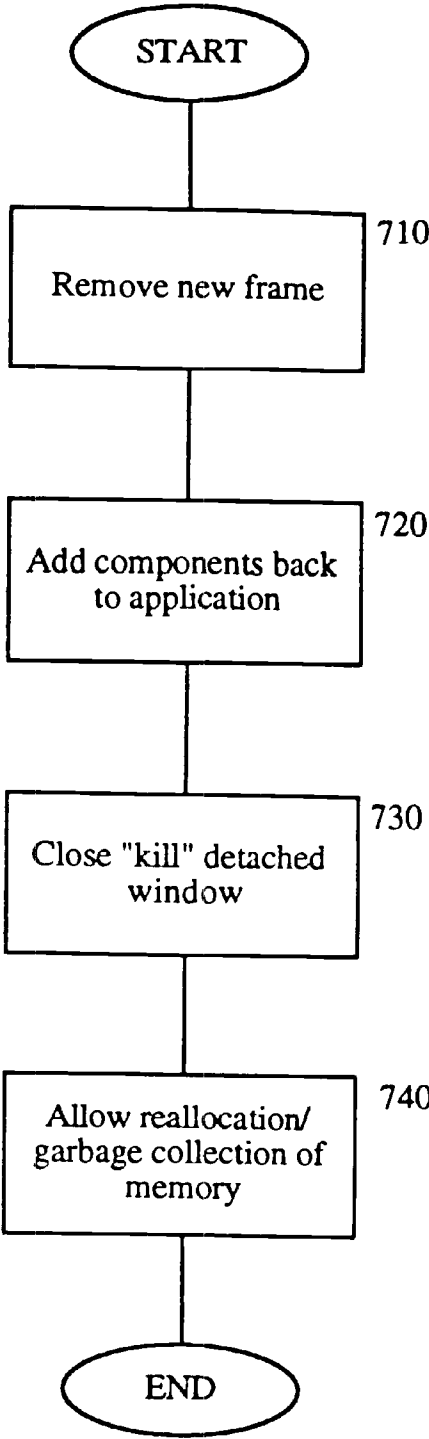


FIGURE 7

DETACHABLE JAVA APPLETS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the field of computer software. More specifically, the present invention relates to applets and their relationship to an operating environment.

2. Description of Related Art

Recent versions of applications such as browsers like Netscape Navigator 3.0™ or Hot Java™ (a product of Sun Microsystems, the Sun logo and Hot Java are trademarks or registered trademarks of Sun Microsystems in the United States and other countries) have provided for the use of platform-independent “applets” which can be downloaded as pre-compiled Java byte-codes (intermediate level instructions). These applets are executed via a “virtual machine,” which is a platform-specific environment that interprets or compiles the applet code and performs the high-level instructions therein. One popular and predominant applet programming language, developed by Sun Microsystems, Inc., is known as “Java™” (a product of Sun Microsystems, the Sun logo and Java are trademarks or registered trademarks of Sun Microsystems in the United States and other countries). Applets programmed in Java or minor variations thereof are referred to as Java applets.

The key usefulness of Java applets is that they are platform-independent, i.e., a Java applet written for platform A will run without modification on platform B provided that both platform A and platform B have virtual machines capable of executing applet code for their respective platforms. Even though Java applets are platform-independent, the characteristics, quirks and limitations of the applications from which they are spawned weaken the flexibility by causing the applets to become essentially “application-dependent.” For example, one limitation of Java applets when called from HTML (Hypertext Markup Language) code for the Sun operating system version of Netscape Navigator is that when applets are called, the HTML tag for the call must include a height and width, thus defining a window size that the applet must execute within. When running inside the application window, the applet is constrained by the stated height and width tag and thus, any output, input, dialog boxes or pop-up windows that are generated for the applet must appear within that constraint.

In this situation, where the applet window is a “sub-window” of the application window, the applet window suffers several impediments. First, the applet window cannot be closed unless the application is quit or until the application transitions to receive data from a new host (in the same window that launched (spawned) the applet). And concomitant with that limitation, when the application that spawned the applet transitions to a new URL (Uniform Resource Locator—the “address” of the host visited by the application) then the applet window closes and the applet ceases execution. The cessation of the applet is out of the control of the user. In certain instances, it is desirable to continue running the applet even though the application has transitioned to a different URL. For instance, a user may desire a streaming audio applet that plays content from an external or remote source which is launched from URL A to continue playing even though the application has proceeded to URL B, which does not have the same applet. Under current practice, it would be necessary to open or spawn a new instance of the application (i.e., open a new application window) to receive its content from URL B so that the other application instance continues to play the audio applet. But this approach suffers from several maladies.

First, launching a new application instance may involve an increase in memory and system resource utilization which will diminish the performance of both the applet and the new application instance. Further, the applet still cannot be controlled outside of the constraints or environment of the application. In fact, with a second application window (instance) launched, the first window must become active (in the foreground, under control of cursor or mouse) before the applet can be controlled. Further, the traditional applet model does not allow for iconification of the applet window within the operating environment (minimizing of the window). Under current practice, the application window itself must be minimized in order to minimize and iconify the applet. In that case, the applet rather than having its own icon, will inherit the icon of the browser. The lack of window minimization, resizing and other GUI modification such as changing fonts, backgrounds colors, etc., imposes severe constraints on the applet to be independently controlled without application constraints.

One solution to remove the application dependence of executable code modules has been the use of “plug-ins”. However, unlike “plug-ins” (file(s) containing data/code used to alter, enhance, or extend the operation of a parent application) that are operating environment/platform specific, Java applets are essentially platform independent. Plug-ins, which must be downloaded (or come packaged with the application), allow certain file types (such as Shockwave™ or RealAudio™) which may not be supported internally by the application to be interpreted and output on the local platform. However, plug-ins are disadvantageous in that they remain resident locally and must be stored on local disk for re-use, unlike the virtual machine of Java which is application resident. Importantly, plug-ins spawn processes wholly independent of the browser and are thus, platform dependent. Thus, though plug-ins may allow for independent GUI control of their windows, they are completely disinherited from the browser unlike Java applets (because they do not require a virtual machine to run). Running a plug-in is akin to running a separate application via the operating environment, and thus is not a viable substitute for portable executability as is a Java applet.

Yet another development for enhancing capabilities of an application such as a browser is the use of “helper” applications. Helper applications, which are stored locally, do not have the portability and platform independence of Java applets, i.e., a helper application on a Pentium platform cannot be used on a Sun Sparc™ (a product of Sun Microsystems, the Sun logo and Sparc are trademarks or registered trademarks of Sun Microsystems in the United States and other countries) system or vice-versa. The helper application also spawns a new process/thread within the operating environment and commands the system resources of a new application instance which is unlike Java applets. The helper application is not related to the application delivering the data to be processed and is merely called through the operating environment. The helper application does not plug-in or execute within a virtual machine of the application. Further, a helper application is not easily transferred from host to client, since helper applications can be quite large in code size when compared to applets.

Further, on newer information devices such as network computers (NCs), helper applications and plug-ins may not even work due to limited operating environment features and lack of local storage. NCs are conceptually based on utilizing remotely stored applets, such as Java applets which are network distributed, to provide applications and content to the NC. In contrast, the current industry standard for NCs

guarantees that NCs are able to execute Java applets, through the use of virtual machine and browser/application. Even in the NC situation, it is desirable that the applet have its own built-in functionality separate from the browser/application from which it is called.

Thus, there is a need for a method and apparatus to detach Java applets from the constraints of the application so that they can be GUI-controlled directly through the operating environment and so that they not be limited by the state of the application in which the applets are spawned.

SUMMARY

A method and system is disclosed for detaching Java applets from the constraints of the application which provides the Java virtual machine for executing those applets. Thus, the applets, when detached, can appear in a detached window which is more easily controllable by the operating environment desktop. The Java applets continue to run under the application's virtual machine but do so with less constraints than the graphical interface limits of the application. Further, if the application that launched the applet transitions to a new URL host, the Java applet continues to run. Also, the applet, once detached, can be reattached into the application to appear in an application history.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a flow diagram of transforming a non-detachable Java applet to have the functionality of detachability according to an embodiment of the invention.

FIG. 2 illustrates a flow diagram of applet behavior when launched from an application according to an embodiment of the invention.

FIG. 3 is a diagram of exemplary inheritance hierarchy as defined for Java applets.

FIG. 4 is an illustration of a resulting detached applet as rendered on a display screen.

FIG. 5 shows an applet "Jukebox" in the attached state.

FIG. 6 is a flow diagram of the detaching of an applet according to one embodiment of the invention.

FIG. 7 is a flow diagram of the attaching of an applet according to one embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

Definitions

The word "operating environment" refers to an operating system, kernel, shell, or the like which is low-level software which schedules tasks, allocates storage, handles the interface to peripheral hardware and presents a default interface to the user when no application program is running. The operating environment also allows application programs to run by performing storage allocation, input/output, etc., that the application may request through its executable code.

The word "desktop" refers to the visual environment (usually output on a display screen) wherein the operating environment allows a user to interact with the operating environment. A desktop includes but is not limited to the display and use of icons, cursors, windows, dialog boxes, menus and other user interface elements which are a function of the operating environment. The desktop is a visually rendered screen region wherein application windows and their associated input/output may be displayed and interacted upon by a user, if desired.

A "method" as used in the detailed description refers to a function call, procedure or routine associated with one or

more classes, which are well-known in the art of object-oriented programming.

The word "detachable" refers to the ability of an applet to become free of GUI constraints imposed upon it by the application that spawned the applet. "Detachability" implies that the applet is capable of being manipulated in a graphical user interface sense separate of the application that spawned it and can instead be manipulated on the desktop by interaction directly with the operating environment.

The word "virtual machine" refers to an interpreter, compiler, linker, etc., which is included in an application or operating environment to facilitate the execution of instructions, pseudo-code and the like for a particular programming language such as Java.

FIG. 1 is a flow diagram of transforming a non-detachable Java applet to have the functionality of detachability according to an embodiment of the invention.

Any pre-existing Java applet can be modified to become a detachable Java applet as detachable is described with respect to various embodiments of the invention. When an applet is defined/created, its source code can be modified to include methods for detaching the applet. The first step is to add an "implements Detachable" statement to the class definition of the applet (step 110). This implements an interface called "Detachable". Appendix A, the Java source code for the Jukebox streaming audio applet, shows on column 2, the class definition "public class Jukebox extends Applet implements Detachable." The phrase "public class <Applet Name> extends Applet" is shared by all applets in their main class definition. The phrase "implements Detachable" may then be appended to any such definition to begin the modification of the <Applet Name> applet to become detachable. Several more steps are desirable in order to complete the foundation for detachability of the applet. The Detachable interface invoked via the class definition is implemented by adding several generic "methods" (see Definitions, above) to the source code of the applet (step 120). The methods while generic in terms of the Detachable Interface are peculiar and unique to this embodiment of the invention in that these methods have not been previously defined in the art of Java development.

The generic methods will allow a Java applet to become detachable from the application in which they were spawned and will allow the applet to have the functionality of any ordinary application window running on the operating environment (e.g., Solaris™, a product of Sun Microsystems, Solaris is a trademark or registered trademark of Sun Microsystems in the U.S. and foreign countries) desktop. The generic methods, if they are not adequate for that specific applet's look-and-feel (checked at step 130), may be modified. The generic methods can be modified to include interfacing that suits the look-and-feel or peculiarities of the applet (step 140). For example, it may be desired in a detachable "chat" (text-based conversation between users) applet that the chat window resize itself to display long strings of text which without resizing itself were invisible. The methods, which are defined below may be modified by one of ordinary skill in the art to suit the applet being transformed into a detachable applet. One such modification, described below, is the addition of "controller" methods. As described with respect to step 120, these generic methods are added to the exemplary Jukebox applet code disclosed in Appendix A.

Generic Detachable Interface Methods and Modifications Thereto

1. "Detach" (illustrated as "public void detach ( ) {" in Appendix A).

The Detach method is the primary function call allowing the applet to be detached from the application. The state variable "isDetached" is set equal to true to indicate the applet is now in the detached as opposed to attached state. The statement "remove(UIpanel)" is responsible for removing the "panel" for the user interface which contains components like user interface elements (i.e., buttons, text boxes, menus, icons, etc.). The panel UIpanel is removed and passed onto the "Detached Frame" class that is instantiated. The "Detached Frame" class is a platform-independent implementation based on the standard Java virtual machine and is thus available to all Java platforms. Code for the Detached Frame class is shown in Appendix B. The Detached Frame is instantiated to create a detached window (application independent applet window) into which UIpanel components (user interface elements) can be rendered. The statement "controller.makeAttachable( )," is an example of a modification of the generic Detach method that is specific to the Jukebox applet that creates a user interface element for attaching the applet back to the application.

2. "Attach" (illustrated as "public void attach ( ) {" in Appendix A)

The Attach method is used to re-attach the applet back to the application and to close the detached window residing outside of the application's window. The state of the applet is returned to attached by setting isDetached equal to false. Next, the Detached Frame object is disposed of. The UIpanel components are then added back to the application (add (UIpanel) statement). An example of applet-specific (in this case, the Jukebox applet) modification to the generic method Attach is the controller.makeDetachable statement which when included provides user interface elements for detaching out the applet.

3. Close (illustrated as "public void close( )" in Appendix A)

The Close method kills the Detached Frame window and typically, the applet does not thereafter re-attach to the application. This method is useful when total purging of the applet is desired.

These generic methods, when added to implement the Detachable interface, will cause a non-detachable applet to become detachable (have the functionality of detaching from the application window).

FIG. 2 illustrates a flow diagram of applet behavior when launched from an application according to an embodiment of the invention.

Though FIG. 2 is directed toward detaching from a application, the methodology is equally applicable to detaching applets from any environment that executes the applet code through a virtual machine.

According to step 210, first, the applet must be launched in the virtual machine of the application. This is usually accomplished automatically when a user visit a web site (HTML page) which includes statement for launching the applet. Assuming that the applet has been transformed according to the methodology of FIG. 1, or if the applet is already capable of detaching, the virtual machine is checking (waiting) for the user to activate the user element, such as a button, to detach the applet from the application (checked at step 220). If the user has activated the "detach" user interface element, then the environment calls the "Detach" method (step 230).

The calling of the "Detach" method, for example, leads to the execution of other instructions as mentioned above. One key result of these instructions is the displaying of a detached window in which the applet controls and perhaps,

data would be rendered on to (step 240). Two checks are continuously being performed once the display has rendered the detached window. The first is to see whether the user changed properties (such as size, background color, etc.) of the detached window (step 255). This step is a check performed not from the application, but from the operating environment itself. If any changes are requested of the properties of the detached window, the operating environment initiates and completes those changed properties (step 250). Such changes include resizing the detaching window, changing the fonts of the window, and since the detached window is a window of the operating environment, minimizing and iconification can be performed without reference, modification of the application window. Further, the applet in the detached window is no longer application constrained, and has its own set of graphical properties—color, background, font, size, etc.—apart from the application. The desktop can control the look-and-feel of the detached window, and consequently, to some degree, the applet as well.

The second check being performed is to query if the user activated the user interface element to attach the applet back to application (step 260). Once the applet is in the Detached Frame, the user interface element (button, etc.) for detaching will be replaced by a user interface element for attaching the applet back to the application. A request by the user to attach the applet to the application will first cause the "Attach" method to be called (step 270). The Attach method includes several instructions as described above but has the primary functionality of closing the detached window (step 280) and removing it from the operating environment. The applet is then redrawn into the application window as when the applet was launched maintaining not the detached look-and-feel, but reverting to the look-and-feel of the application which launched the applet. The applet, when redrawn into the application will replace the user interface element for attach with the user interface element for detach. Thus, the applet can switch states from attached to detached as the user so desires. Not shown in FIG. 2 is the case where the application has transitioned to a new host prior to the applet being attached. In that case, the applet, rather than being redrawn in the application window, becomes part of the application's "history". The history is a record of previously visited URLs so that users of the application can return to those URL sites again.

FIG. 3 is a diagram of exemplary inheritance hierarchy as defined for Java applets.

From object or class inheritance viewpoint, there are two sets of inheritance trees for the various embodiments of the invention. The first, shown on the right-hand side of FIG. 3 is the generic Java applet inheritance hierarchy. An Object 310 is inherited into a Component 320. The Component 320 (and consequently Object 310) is inherited into Container 330. Container 330 is inherited into Panel 340. Finally, Panel 340 is inherited into Applet 350.

The second hierarchy shown in FIG. 3 is the hierarchy created by inheritance of the Detached Frame. This hierarchy is a representation of the methodology of detaching an applet described with respect to various embodiments of the invention.

Like the generic applet inheritance hierarchy, the left side of FIG. 3, the Detached Frame inheritance hierarchy shows an Object 315 inherited into a Component 325. A Container 335 is, likewise, inherited into a Container 335. FIG. 3 shows that the two hierarchies are identical up to and including the Container object. Thus, the hierarchies may be joined at Containers 335 and 330. The joining of the



inheritance hierarchy is conceptually the object-oriented mechanism allowing an Applet 350 to be transferred from the application window to the detached frame since both are instances of class container. One skilled in the art of object-oriented programming will readily be able to utilize the property of joined object hierarchies described above to implement the various embodiments of the invention.

FIG. 4 is an illustration of a resulting detached applet as rendered on a display screen.

FIG. 4 shows a display 400 which may any monitor, display or other visual device that renders a application window 410 through the operating environment. Application window 410 is shown as an application window which contains several elements such as a picture and URL (Uniform Resource Location) (labeled as "location"), but may be any window or display environment running over the operating environment. Display 400 shows the application window 410 running over an operating environment desktop 430 (see above for definition of "desktop"). The operating environment desktop 430 and application window 410 may be more intimately or tightly integrated such as in Hot Java Views™ (a product of Sun Microsystems, the Sun logo and Hot Java views are trademarks or registered trademarks of Sun Microsystems in the United States and other countries). The invention can be modified as to remove any lingering visual and user interface constraints of the application window which may restrain an applet however tightly integrated the application and environment may seem. In this sense, FIG. 4 shows an applet named "Jukebox" in a detached window 420. The applet was launched at some URL location and initially, contained within application window 410 (see FIG. 5). FIG. 4 shows the detached state of the applet Jukebox.

In this detached state, the applet controls "About", "Stop" and "Play" are rendered into the detached window. An important aspect to the invention is the ability of the applet to continue running or executing its instructions under the virtual machine (interpreter) of the application whose window is rendered in 410. The operating environment and its desktop 430 now controls the general look-and-feel of the window for the applet Jukebox. The detached window can then be manipulated like any other window on the desktop. The application window 410 and the interface properties of the application no longer control, constrain or limit the GUI characteristics of the applet. Further, when the application window 410 transitions to a new host URL, the applet continues to run in the Detached Frame. Though the virtual machine is platform-dependent, i.e., it must decompose Java code into processor/platform native code, the Java applet and its code is not. When the application closes all of its windows completely, the detached applet is closed as it should be. Further, the applet, even though detached, must cease execution because it is no longer streaming data from the host that was contacted by the application. Thus, the application retains control of concurrently terminating the applet when terminating itself even though the applet is detached from its windowing and interface constraints of the application window.

While in the detached state, several other GUI modifications are available to the detached window which are not explicitly illustrated. First, unlike traditional applets, a detached window 420 can be iconified onto an area of the desktop 430 or minimized into a toolbar or other GUI element without having to iconify the application window 410 as well. Thus, the minimized applet can have its own icon. Further, the detached window 420 may be resized on the desktop 430 while in the detached state without refer-

ence to the height-width tag within the HTML (Hypertext Markup Language) or other document which specified the calling of the applet from within the application window 410. This allows the detached applet greater flexibility in its appearance than the non-detachable applet. Further modifications such as a change of font type, font size, color, background, etc., that are available to other windows running on desktop 430 are also available for the detached window 420 and the applet it displays.

The detached window 420 contains, in addition to applet control for the Jukebox, a "control" (a user interface element) is rendered called "Attach" which, when activated, will close the detached window 420 and collapse the applet back into application window 410. This "Attached" state is shown in FIG. 5. If application window had transitioned to a new host URL, then the applet is included in the application history instead of being instantly executed in the application window.

FIG. 5 shows an applet "Jukebox" in the attached state.

FIG. 5 also has a display device 500, an operating environment desktop 530 and a application window 510. When an applet is in the attached state, the application window 410 constrains it. However, if the applet has been made detachable, a "detach" control (button) is rendered into the application window in the panel of the applet so that the attached state may be modified to the detached state leading to a detached window as shown in FIG. 4. Alternatively, the applet can be launched automatically into the detached state with a control for attaching as shown in FIG. 4. A "toggle" method can also be provided to toggle the state of the applet regardless of what the current state is. The toggle can be initiated by a user interface element in both the detached window when the applet is in the detached state and the applet panel (in the application window) when the applet is in the attached state. The toggle method may be desirable where a uniform control is desired in both attached and detached states. The attached state shown in FIG. 5 is unlike the prior art since the applet shown is "detachable" via the "detach control." An applet without detachability would be limited to the application itself and users would be usable to change the display properties of the applet Panel 550 in the application window 510. A detachable applet in the attached state suffers the same limitation, but can be freed by activating "Detach." The attached applet state of FIG. 5 assures that prior to being re-attached from a "detached" state (shown in FIG. 4), the host URL of the application window 510 had not changed.

FIG. 6 is a flow diagram of the detaching of an applet according to one embodiment of the invention.

The method definition of "Detach" described above with respect to code Appendix A is not fully represented in FIG. 6 but it will be understood by one skilled in the art that any additional steps shown or omitted from the method definition can easily be implemented.

The first step in detaching is to instantiate a new frame called Detached Frame (see FIG. 1 description of Detach method) (step 610). The new frame is then used as a drop-off point for applet data, content and user interface elements/controls. Next, the applet components are removed from the application applet panel (step 620). These components are objects including, but not limited to, buttons, controls, user interface elements, action messages, dialog boxes, data, etc., which may be renderable to the display device. The applet components are then added to the user interface of the new frame (step 630). Additionally, an "attach" user interface element is added to the new frame in addition to other components so that the applet may re-attach to the applica-

tion (step 640). Next, the detached window generated through the new frame is mapped or rendered on the display device (step 645). The detached window is, at this stage of the process, a blank window capable of interfacing within the desktop environment of the operating environment. As implemented in various embodiments of the invention, the detached window will inherit size, font, color characteristics of the applet panel as attached in the application window (step 655). Finally, the components, including the attach control, are painted on the screen using the "Paint All" Java method (step 660). The "Paint All" method, well-known in the art of Java programming, is used to render objects to the display or operating environment, rather than within a portion of the application window. This ensures that components for the detached applet will be properly rendered, i.e., without showing up as meshed with other screen graphics or as hidden from view.

FIG. 7 is a flow diagram of the attaching of an applet according to one embodiment of the invention.

Again, though the flow diagram of FIG. 7 may vary from the method "Attach" described earlier, one skilled in the art

will readily be able to exchange/add features of that method into the flow diagram of FIG. 7 and vice-versa.

When the attach control is activated, first the new frame instantiated by the calling of the Detach method is disposed from the virtual machine environment (step 710). Components which were removed from the application environment are added back to the application, and automatically rendered in a panel therein as with a typical Java applet (step 720). Next, the detached window is closed (step 730) or removed from the operating environment. Finally, garbage collection and reallocation of memory (step 740) may be allowed to recycle the resources utilized by the detached frame instantiated by the Detach method back to the operating environment.

While the present invention has been particularly described with reference to the various figures, it should be understood that the figures are for illustration only and should not be taken as limiting the scope of the invention. Many changes and modifications may be made to the invention, by one having ordinary skill in the art, without departing from the spirit and scope of the invention.



US 6,401,134 B1

11

12

APPENDIX A

Jukebox.java

2

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633

```

US 6,401,134 B1

13

14

APPENDIX A

Jukebox.java

```
3
/*
 * Used when the Jukebox is invoked as an applet
 */
public void init() {
    HTMLConfig();
    docbase = getDocumentBase();
    codebase = getCodeBase();
    isDetachable = true;
    initialize();
}

/*
 * Used when the Jukebox is invoked as a standalone application
 */
public static void main(String argv[]) {
    if (argv.length < 1) {
        System.out.println("Usage: java Jukebox <url> [<files>"];
        System.exit(1);
    }

    Jukebox jukebox = new Jukebox();
    try {
        jukebox.docbase = new URL(argv[0]);
        jukebox.codebase = null;
        jukebox.URLConfig(jukebox.docbase);
    } catch (MalformedURLException e) {
        jukebox.selection = "";
        for (int i = 0; i < argv.length; i++)
            jukebox.selection += argv[i] + " ";
        jukebox.mediaType = "file";
        jukebox.showslides = false;
    }

    jukebox.isDetachable = false;
    Jukebox.initialize();

    new DetachedFrame("Java Jukebox", jukebox,
        jukebox.OptionPane, jukebox.IconImage);
}

/*
 * Get jukebox parameters from the applet: html tag
 */
private void HTMLConfig() {
    /*
     * Get the server name and port number to connect to.
     */
    server = getParameter("server");
    try {
        port = Integer.parseInt(getParameter("port"));
    } catch (Exception e) {}

    /*
     * Get selection list and associated image names
     * followed by appearance parameters
     */
    mediaDir = getParameter("mediadir");
    mediaType = getParameter("mediatype");
    slideDir = getParameter("slidedir");
    selection = getParameter("selection");
    foreground = getParameter("foreground");
    background = getParameter("background");
    fontName = getParameter("fontName");
    fontStyle = getParameter("fontstyle");
    if (getParameter("showslides") != null)
        showSlides = getParameter("showslides").equals("true");
    try {
        fontSize = Integer.parseInt(getParameter("fontsize"));
    } catch (Exception e) {}
    try {
        slideHeight = Integer.parseInt(
            getParameter("slideheight"));
    } catch (Exception e) {}
    try {
        slideWidth = Integer.parseInt(
            getParameter("slidewidth"));
    } catch (Exception e) {}

    String config = null;
    if ((config = getParameter("config")) != null) {
        try {
            URLConfig(new URL(getDocumentBase(), config));
        } catch (MalformedURLException e) {
            System.out.println("Unable to resolve " + config);
            System.out.println("Jukebox not configured");
        }
    }

    /*
     * Get jukebox parameters from a URL
     */
    private void URLConfig(URL configURL) {
        if (configURL == null)
            return;

        StringTokenizer t1, t2;
        String text = "";
        Hashtable params = new Hashtable();

        try {
            DataInputStream s;
            String inputLine;

            s = new DataInputStream(configURL.openStream());
            while ((inputLine = s.readLine()) != null) {
                text = text + inputLine;
            }
            s.close();
        } catch (Exception e) {
            System.out.println("Error loading config URL: " + e);
            e.printStackTrace();
            return;
        }

        t1 = new StringTokenizer(text, "\\");
        while (t1.countTokens() >= 2) {
            String arg = t1.nextToken();
            String value = t1.nextToken();

            t2 = new StringTokenizer(arg, "\\");
            String paramName = t2.nextToken();
            params.put(paramName, value);

            if (params.containsKey("server"))
                server = (String)params.get("server");
            if (params.containsKey("port"))
                try {}
        }
    }
}
```

US 6,401,134 B1

15

16

2008-12-10 14:00:00 APPENDIX A

Jukebox.java

5

```
port = Integer.parseInt((String)params.get("port"));
    } catch (Exception e) {
        mediadir = (String)params.get("mediadir");
        if (params.containsKey("mediatype"))
            mediatype = (String)params.get("mediatype");
        if (params.containsKey("slidedir"))
            slidedir = (String)params.get("slidedir");
        if (params.containsKey("showslides")) {
            String val = (String)params.get("showslides");
            showslides = val.equals("true");
        }
        if (params.containsKey("selection"))
            selection = (String)params.get("selection");
        if (params.containsKey("foreground"))
            foreground = (String)params.get("foreground");
        if (params.containsKey("background"))
            background = (String)params.get("background");
        if (params.containsKey("fontname"))
            fontname = (String)params.get("fontname");
        if (params.containsKey("fontstyle"))
            fontstyle = (String)params.get("fontstyle");
        if (params.containsKey("fontsize"))
            try {
                fontsize = Integer.parseInt(
                    (String)params.get("fontsize"));
            } catch (Exception e) {
            }
        if (params.containsKey("slidewidth"))
            try {
                slidewidth = Integer.parseInt(
                    (String)params.get("slidewidth"));
            } catch (Exception e) {
            }
        if (params.containsKey("slideheight"))
            try {
                slideheight = Integer.parseInt(
                    (String)params.get("slideheight"));
            } catch (Exception e) {
            }
        if (params.containsKey("slideshow"))
            try {
                slideshow = Integer.parseInt(
                    (String)params.get("slideshow"));
            } catch (Exception e) {
            }
    }

private void initialize() {
    String title = null;
    Toolkit tk = getToolkit();

    /* Set some defaults for missing parameters
    */
    if (mediadir == null)
        mediadir = "";
    if (mediatype == null)
        mediatype = "net";
    if (slidedir == null)
        slidedir = "";
    if (fontname == null)
        fontname = "TimesRoman";
    if (fontstyle == null)
        fontstyle = "plain";
    if (server == null)
        if (docbase != null)
            server = docbase.getHost();
        else
            server = "localhost";
    if (fontsize == 0)
        fontsize = 12;
    if (slidewidth == 0)
```

6

```
        slideheight = 200;
    if (slidewidth == 0)
        slidewidth = 200;
    if (port == 0) {
        if (mediatype.equals("live") ||
            mediatype.equals("liveURL") ||
            port == LIVEAUDIO_PORT)
            port = Jukebox_PORT;
        else if (mediatype.equals("net"))
            port = Jukebox_PORT;
    }

    /*
    * The selection is in a string of the format
    * audio_file:title_string:image_file
    * or:
    * audio_file:title_string
    * or:
    * title_string
    * depending on the media type.
    * newlines, tabs, and spaces are ignored.
    */
    StringTokenizer t = new StringTokenizer(selection, "\\n\\t");
    if (mediatype.equals("live") || mediatype.equals("liveURL")) {
        title = t.nextToken();
    } else {
        tracks = new String[t.countTokens()];
        slides = new Image[t.countTokens()];
        titles = new String[t.countTokens()];
        for (int i = 0; i < t.countTokens(); i++) {
            String triple = t.nextToken().trim();
            int j = triple.indexOf(':');
            if (j == -1) {
                // no title or image was given
                tracks[i] = triple;
                titles[i] = triple;
                slides[i] = null;
                continue;
            }
            tracks[i] = triple.substring(0, j);
            int k = triple.indexOf(':', j+1);
            if (k == -1) {
                // no image was given
                titles[i] = triple.substring(j+1);
                slides[i] = null;
                continue;
            }
            titles[i] = triple.substring(j+1, k);
            try {
                slides[i] = tk.getImage(
                    new URL(docbase, slidedir+
                        triple.substring(k+1)));
            } catch (MalformedURLException e) {
                System.out.println("Unable to get: "+
                    slidedir+triple.substring(k+1));
            }
        }
    }
    try {
```

US 6,401,134 B1

17

8

Jukebox.java

```

aboutimg = tk.getImage(new
    URLEncoder, "StreamInfo.class");
iconimg = tk.getImage(new
    URLEncoder, "StreamHeader.class");
} catch (MalformedURLException e) {
    aboutimg = null;
    iconimg = null;
}

/*
 * Construct the Selection box using the given
 * parameters
 */
if (mediatype.equals("live") || mediatype.equals("liveurl")) {
    selbox = new JukeboxSelection(title);
} else {
    if (showslides)
        selbox = new JukeboxSelection(mediadir, tracks,
            titles, slides,
            slidewidth, slideheight);
    else
        selbox = new JukeboxSelection(mediadir, tracks,
            titles);
}

/*
 * Construct the controller and pass the selection box
 * to it.
 */
controller = new JukeboxControl(server, port, selbox,
    mediatype, this, docbase,
    isdetachable, aboutimg);

/*
 * Tell the selection box who is observing it
 */
selbox.setobserver(controller);

/*
 * Layout the GUI components. AWT sucks.
 */
uiPanel = new JPanel();
uiPanel.setLayout(new BorderLayout());
setColors(uiPanel, background, foreground);
setColors(this, background, foreground);
setFont(uiPanel, fontName, fontStyle, fontSize);
uiPanel.add("North", selbox);
uiPanel.add("South", controller);
add(uiPanel);

public void start() {
    if (!isDetached)
        controller.resume();
}

public void stop() {
    if (!isDetached)
        controller.pause();
}

public void destroy() {
    if (!isDetached)
        detachedFrame.close();
    controller.abort();
}

/*
 * Detach the GUI and bring it up in a separate window
 * so that the user can control the jukebox separately
 * from the browser.
 */
public void detach() {
    isdetached = true;
    remove(uiPanel);
    detachedFrame = new DetachedFrame("Java Jukebox", this,
        uiPanel, iconimg);
    controller.makeAttachable();
}

/*
 * Attach the GUI back to the page in the browser.
 */
public void attach() {
    isdetached = false;
    detachedFrame.close();
    add(uiPanel);
    controller.makeDetachable();
    paintAll(getGraphics());
}

/*
 * stop the audio and
 * attach the GUI back to the page in the browser.
 */
public void close() {
    controller.stop();
    attach();
}

public boolean isDetached() {
    return isdetached;
}

private void setColors(Panel p, String background, String foreground) {
    Color c;
    c = getColor(background);
    if (c != null) {
        p.setBackground(c);
    }
    c = getColor(foreground);
    if (c != null) {
        p.setForeground(c);
    }
}

private Color getColor(String name) {
    if (name == null)
        return null;
    if (name.equalsIgnoreCase("black"))
        return Color.black;
    else if (name.equalsIgnoreCase("blue"))
        return Color.blue;
    else if (name.equalsIgnoreCase("cyan"))
        return Color.cyan;
    else if (name.equalsIgnoreCase("darkgray"))

```

18

2003223 APPENDIX A 000

Jukebox.java

9

```
        return Color.darkGray;
    else if (name.equalsIgnoreCase("gray"))
        return Color.gray;
    else if (name.equalsIgnoreCase("green"))
        return Color.green;
    else if (name.equalsIgnoreCase("lightGray"))
        return Color.lightGray;
    else if (name.equalsIgnoreCase("magenta"))
        return Color.magenta;
    else if (name.equalsIgnoreCase("orange"))
        return Color.orange;
    else if (name.equalsIgnoreCase("pink"))
        return Color.pink;
    else if (name.equalsIgnoreCase("red"))
        return Color.red;
    else if (name.equalsIgnoreCase("white"))
        return Color.white;
    else if (name.equalsIgnoreCase("yellow"))
        return Color.yellow;
    else {
        StringTokenizer t = new StringTokenizer(name, " ");
        if (t.countTokens() == 3) {
            try {
                int red = Integer.parseInt(t.nextToken());
                int green = Integer.parseInt(t.nextToken());
                int blue = Integer.parseInt(t.nextToken());
                return new Color(red, green, blue);
            } catch (Exception e) {
                return null;
            }
        }
        return null;
    }
}

private void setFonts(Panel p, String name, String style, int size) {
    int fontStyle;
    if (style.equalsIgnoreCase("tallc"))
        fontStyle = Font.ITALIC;
    else if (style.equalsIgnoreCase("bold"))
        fontStyle = Font.BOLD;
    else if (style.equalsIgnoreCase("bolditalic"))
        fontStyle = Font.BOLD + Font.ITALIC;
    else
        fontStyle = Font.PLAIN;
    p.setFont(new Font(name, fontStyle, size));
}
```

US 6,401,134 B1

APPENDIX B  
DetachedFrame.java

1

```
/**
 * Class DetachedFrame
 *
 * A frame for holding a Detachable object in a separate window
 */
import java.awt.*;

public class DetachedFrame extends Frame {
    Detachable detachable;

    public DetachedFrame(String title, Detachable d,
        Component c, Image img) {
        detachable = d;
        setTitle(title);
        if (img != null) {
            Toolkit tk = getToolkit();
            for (int i = 0; i < 4; i++) {
                if (tk.prepareImage(img, -1, -1, this))
                    break;
            }
        } else {
            try {
                Thread.sleep(500);
            } catch (Exception e) {
                break;
            }
        }
    }

    public void setIconImage(Image img) {
        Panel p = new Panel();
        p.add(c);
        p.setBackground(c.getBackground());
        p.setForeground(c.getForeground());
        setLayout(new BorderLayout());
        add("Center", p);
        pack();
        show();
    }

    // properly dispose of the UI components and close the window
    public void close() {
        hide();
        dispose();
    }

    // close the window and abort if the user closes the window
    public boolean handleEvent(Event e) {
        if (e.id == Event.WINDOW_DESTROY) {
            detachable.close();
            hide();
            dispose();
            return true;
        }
        return false;
    }
}
```



What is claimed is:

1. A process comprising:  
implementing a detachable interface to enable an applet to become free of graphical user interface (GUI) constraints imposed by an application that spawned said applet; and  
modifying a non-detachable applet to become a detachable applet, said detachable applet capable of being manipulated on a desktop by interaction with an operating environment.

2. A process according to claim 1 wherein the implementing comprises:  
modifying the class definition of said non-detachable applet to include a detachable interface.

3. A process according to claim 2 wherein the modifying comprises:  
adding a set of generic methods for implementing said detachable interface.

4. A process according to claim 3 further comprising:  
modifying any of said set of generic methods to suit the desired look-and-feel of said detachable applet.

5. A process according to step 3 wherein said adding comprises:  
adding a first method for placing said detachable applet into a detached state; and  
adding a second method for placing said detachable applet if in the detached state back to an attached state.

6. A process according to claim 5 further comprising:  
adding a third method for toggling between said states.

7. A process according to claim 5 further comprising:  
adding a fourth method for disposing completely said detachable applet while in the detached state.

8. A process for controlling the behavior of an applet exclusive of application constraints comprising:  
detaching said applet from said application, said applet continuing to utilize a virtual machine of said application to execute applet instructions;  
displaying a detached window to render visually said applet; and  
enabling the modification of visual properties of said detached window.

9. A process according to claim 8 wherein the detaching comprises:  
activating a user interface element to detach applet while graphically constrained by said application; and  
calling a Detach method for executing a set of instructions pursuant to completing said steps of detaching, displaying and enabling.

10. A process according to claim 8 further comprising attaching said applet into said application.

11. A process according to claim 10 wherein the attaching comprises:  
calling an Attach method;  
closing said detached window; and  
redrawing said applet into said application.

12. A process according to claim 9 wherein the calling a Detach method initiates:  
instantiating a new frame;  
removing components of said applet from said application;  
adding said removed components to said new frame;

mapping said detached window onto a display device; and  
painting said added components onto said display device, said painting to occur within the mapped detached window.

13. A graphical user interface system for controlling an applet comprising:  
a desktop defined by an operating environment;  
an application window visually displayed as an overlay on said desktop, said application window displaying and visually constraining said applet while said applet is in an attached state; and  
a detached window running directly over said desktop, said detached window being displayed on said desktop only while said applet is in a detached state, said detached state removing said applet from being visually constrained and displayed in said application window, wherein said applet is a platform-independent program that is executable by a virtual machine.

14. A graphical user interface system according to claim 13 wherein said detached window includes a user interface element for enabling said applet to be placed in the attached state.

15. A graphical user interface system according to claim 14 wherein said application window includes a user interface element for enabling said applet to be placed in the detached state.

16. A graphical user interface according to claim 14 wherein said detached window is closed when said applet is placed in the attached state.

17. A graphical user interface system according to claim 13 wherein said applet is launched initially in the attached state.

18. A graphical user interface system according to claim 13 wherein said applet is launched initially in the detached state.

19. A computer-readable medium having stored thereon applet code interpretable by an application, said applet code including sequences of instructions which, when executed by a processor, cause said processor to perform:  
detaching said applet from said application, said applet continuing to utilize a virtual machine of said application to facilitate execution of said sequence of instructions by said processor;  
displaying a detached window to render visually said detached applet; and  
enabling the modification of visual properties of said detached window.

20. A computer software product having applet code interpretable by an application, said computer software product distributed to a processor, said applet code including sequences of instructions which, when executed by said processor, cause said processor to perform:  
detaching said applet from said application, said applet continuing to utilize a virtual machine of said application to facilitate execution of said sequence of instructions by said processor;  
displaying a detached window to render visually said detached applet; and  
enabling the modification of visual properties of said detached window.



US006415319B1

(12) **United States Patent**  
**Ambroziak**

(10) **Patent No.:** **US 6,415,319 B1**  
(45) **Date of Patent:** **\*Jul. 2, 2002**

(54) **INTELLIGENT NETWORK BROWSER  
USING INCREMENTAL CONCEPTUAL  
INDEXER**

(75) Inventor: **Jacek R. Ambroziak**, Acton, MA (US)

(73) Assignee: **Sun Microsystems, Inc.**, Palo Alto, CA (US)

(\*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/797,630**

(22) Filed: **Feb. 7, 1997**

(51) **Int. Cl.**<sup>7</sup> ..... **G06F 15/16**

(52) **U.S. Cl.** ..... **709/219; 707/513; 707/103**

(58) **Field of Search** ..... 395/611, 603,  
395/614, 615, 777, 793; 709/202, 203,  
206, 217, 219; 707/103, 513

(56) **References Cited**

**U.S. PATENT DOCUMENTS**

4,839,853	A	6/1989	Deerwester et al.	.....	395/605
4,849,898	A	7/1989	Adi	.....	364/419.1
4,984,178	A	1/1991	Hemphill et al.	.....	395/759

(List continued on next page.)

**FOREIGN PATENT DOCUMENTS**

WO WO 96/12236 4/1996

**OTHER PUBLICATIONS**

Okada et al., Agent-based Approach for Information Gathering on Highly Distributed and Heterogeneous Environment, IEEE Ref. No. 0-8186-7267-6/96, pp. 80-87, Jun. 1996.\*

Hof, This Web Agent Knows What You Like, Business Week, New York, Sep. 23, 1996, p. 142E.\*

Webb, Intelligent agents on the Internet, Editor & Publisher, New York, Mar. 25, 1995, pp. 50ff.\*

Peterson, The evolution of intelligent agents, Business Communications Review, Hinsdale, Nov. 1996, pp. 38-41.\*

Quintana, Y. "Knowledge-Based Information Filtering of Financial Information," Proceedings of the National Online Meeting, May 13, 1997, pp. 279-285.

Obraczka, K., et al., "Internet Resource Discovery Services," Computer, vol. 26, No. 9, Sep 1, 1993, pp. 8-22.

"Conceptual Indexing for Precision Content Retrieval," Knowledge Technology Group, Sun Microsystems Laboratories, Chelmsford, MA, Dec. 13, 1996.

Lieberman, Henry, "Letizia: An Agent That Assists Web Browsing," Media Laboratory, Massachusetts Institute of Technology, Cambridge, MA, Aug. 1995.

"WBI: Configuring Your Browser," IBM Web Browser Intelligence, Raleigh, N.C., Dec. 13, 1996.

Woods, William A. and Schmolze, James G., "The KL-ONE Family," Harvard University, Center for Research in Computing Technology, Aiken Computation Laboratory, Cambridge, MA, Aug. 3, 1990, pp. 1-61.

(List continued on next page.)

*Primary Examiner*—Glenton B. Burgess

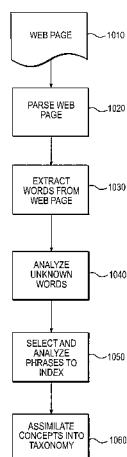
*Assistant Examiner*—Bradley Edelman

(74) *Attorney, Agent, or Firm*—Finnegan, Henderson, Farabow, Garrett & Dunner, LLP

(57) **ABSTRACT**

Network browsing is facilitated by receiving a document from the network containing content; extracting conceptual information from the content of the document; analyzing the extracted conceptual information semantically; and assimilating the extracted conceptual information into an index based on structural relationships among the extracted conceptual information and semantic data in a stored lexicon.

**8 Claims, 9 Drawing Sheets**



US 6,415,319 B1

Page 2

U.S. PATENT DOCUMENTS

5,062,074	A	10/1991	Kleinberger	395/605
5,276,616	A	1/1994	Kuga et al.	364/419.08
5,301,109	A	4/1994	Landauer et al.	364/419.19
5,321,833	A	6/1994	Chang et al.	
5,325,298	A	6/1994	Gallant	364/419.19
5,404,514	A	4/1995	Kageneck et al.	395/605
5,418,918	A	5/1995	Turtle	395/600
5,418,943	A	5/1995	Borgida et al.	
5,418,951	A	5/1995	Damashek	395/605
5,428,778	A	6/1995	Brookes	395/605
5,440,481	A	8/1995	Kostoff et al.	364/419.19
5,450,580	A	9/1995	Takada	395/600
5,475,588	A	12/1995	Schabes et al.	395/2.64
5,544,352	A	8/1996	Egger	395/600
5,708,825	A *	1/1998	Sotomayor	707/501
5,724,571	A *	3/1998	Woods	707/5
5,768,578	A *	6/1998	Kirk et al.	395/611
5,822,539	A *	10/1998	Van Hoff	709/236
5,870,559	A *	2/1999	Leshem et al.	709/224
5,894,554	A *	4/1999	Lowery et al.	709/203
5,918,013	A *	6/1999	Mighdoll et al.	709/217
5,918,237	A *	6/1999	Montalbano	707/513
5,937,163	A *	8/1999	Lee et al.	709/218
6,026,409	A *	2/2000	Blumenthal	707/104
6,032,196	A *	2/2000	Monier	709/245
6,038,561	A *	3/2000	Snyder et al.	707/6
6,081,829	A *	6/2000	Sidana	709/203
6,101,491	A *	8/2000	Woods	707/3
6,263,335	B1 *	7/2001	Paik et al.	707/5

OTHER PUBLICATIONS

Woods, William A., "Understanding Subsumption and Taxonomy: A Framework for Progress," Harvard University, Center for Research in Computing Technology, Aiken Computation Laboratory, Cambridge, MA, Aug. 15, 1990, pp. 1-61.

O'Leary, Daniel E., "The Internet, Intranets, and the AI Renaissance," IEEE Computer, Jan. 1997, pp. 71-78.

Salton, G., et al., "Approaches to Passage Retrieval in Full Text Information Systems," Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 93), ACM Press, 1993, pp. 49-58.

Callan, J.P., "Passage-level Evidence in Document Retrieval," Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGIR 94), Springer-Verlag, 1994, pp. 302-310.

Wilkinson, R., "Effective Retrieval of Structure Documents," Proceedings of the Seventeenth Annual International ACM-SIGR Conference on Research and Development in Information Retrieval (SIGIR 94), Springer-Verlag, 1994, pp. 311-317.

Eric Brill, "Some Advances in Transformation-Based Part of Speech Tagging," AAAI Conference, 1994.

Mittendorf, E., et al., "Document and Passage Retrieval Based on Hidden Markov Models," Proceedings of the Seventh Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGR 94), Springer-Verlag, 1994, pp. 318-327.

\* cited by examiner

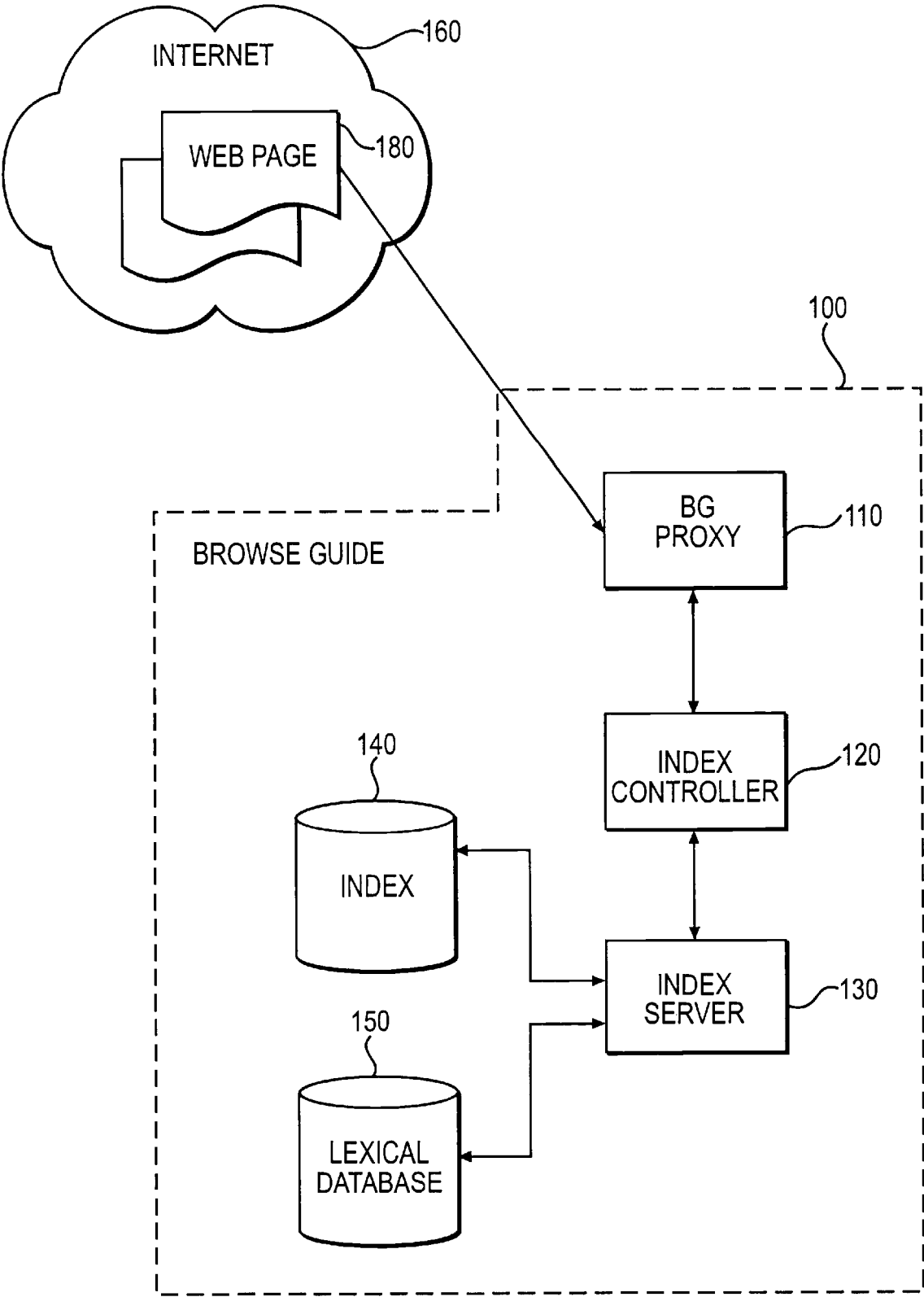
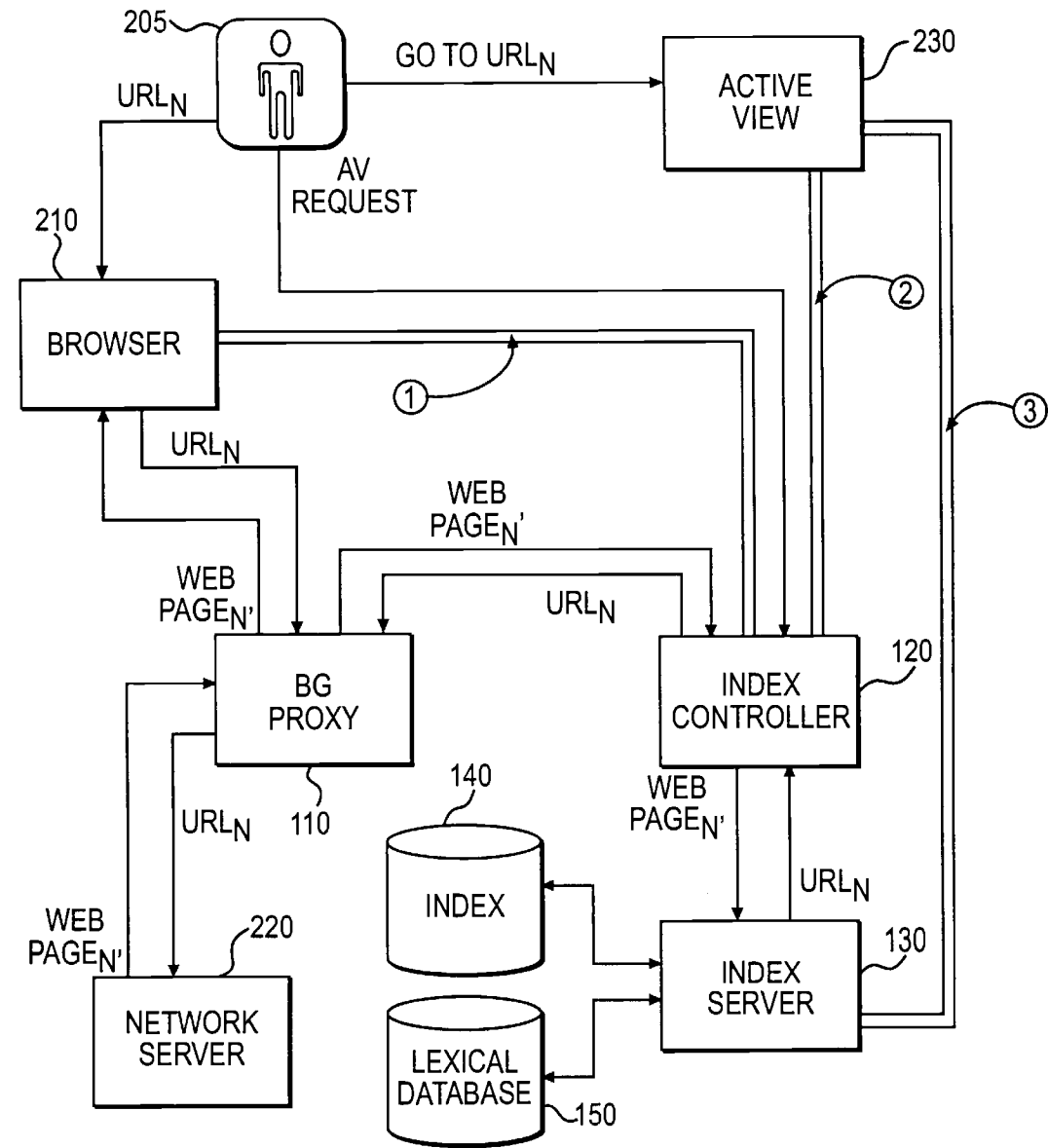
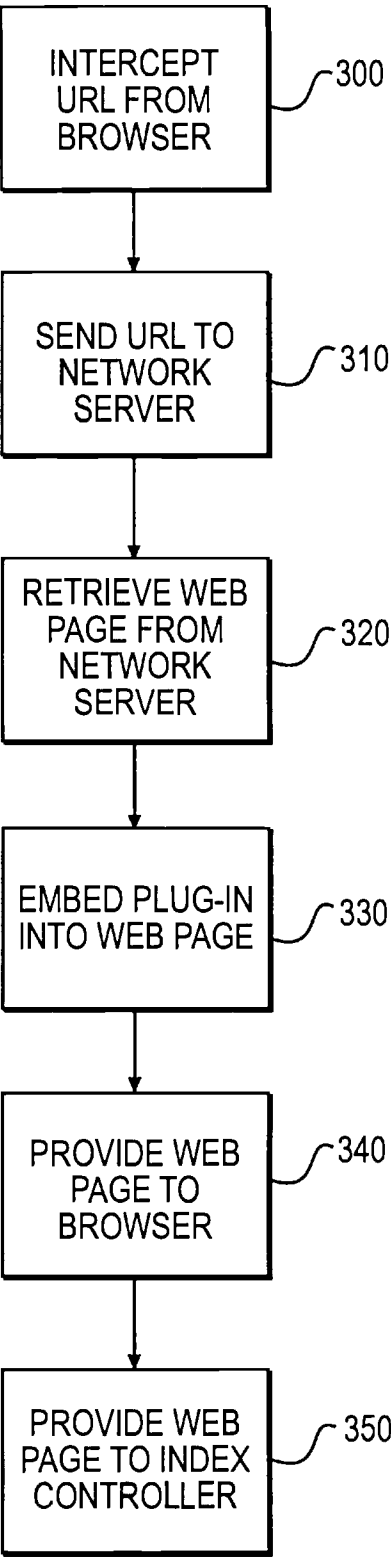


FIG. 1

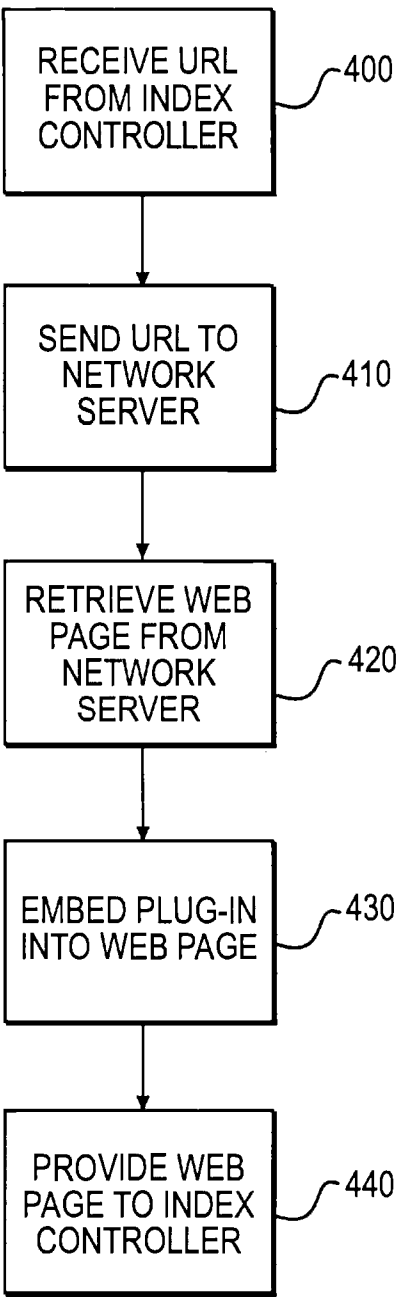


	FROM	TO	ACTION
①	BROWSER	INDEX CONTROLLER	PAGE NOTIFICATION FOR QUEUE PRIORITY
	INDEX CONTROLLER	BROWSER	GO TO URL
②	INDEX CONTROLLER	ACTIVE VIEW	CREATE VIEW AND LINK
	ACTIVE VIEW	INDEX CONTROLLER	GO TO URL
③	ACTIVE VIEW	INDEX SERVER	CONNECT
	INDEX SERVER	ACTIVE VIEW	UPDATES

FIG. 2



**FIG. 3**



**FIG. 4**

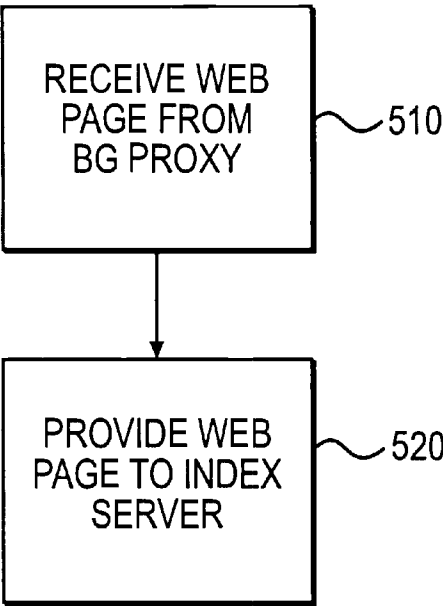


FIG. 5

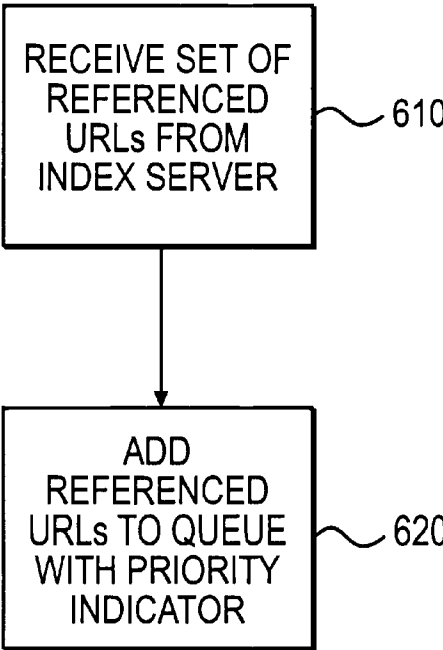


FIG. 6

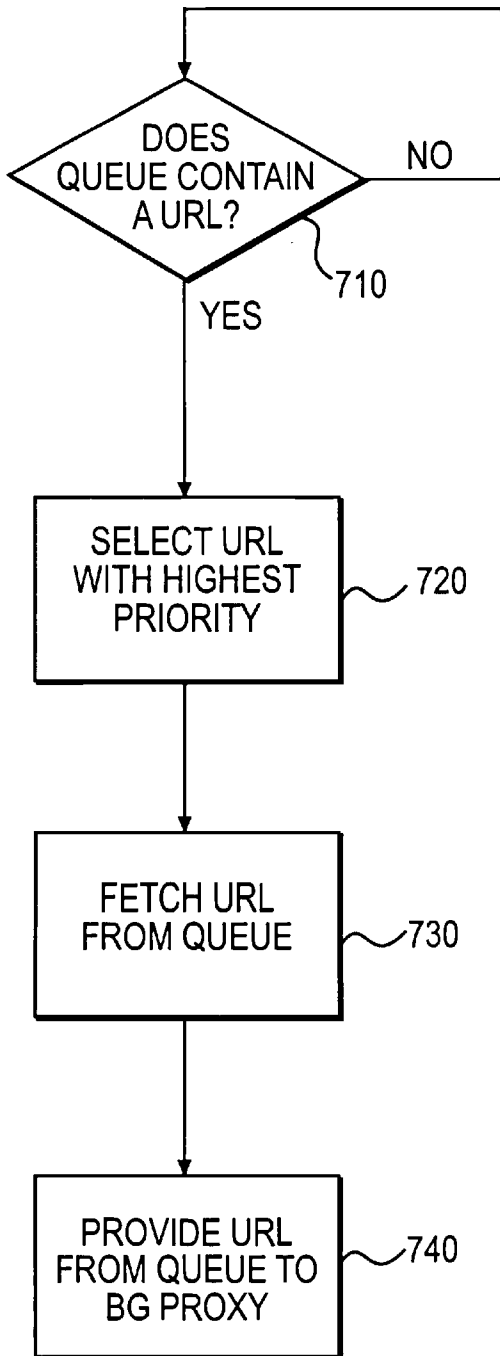


U.S. Patent

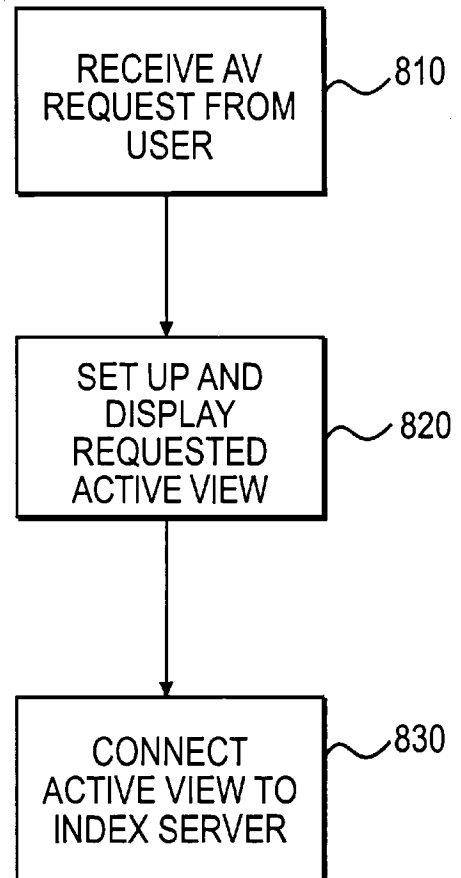
Jul. 2, 2002

Sheet 5 of 9

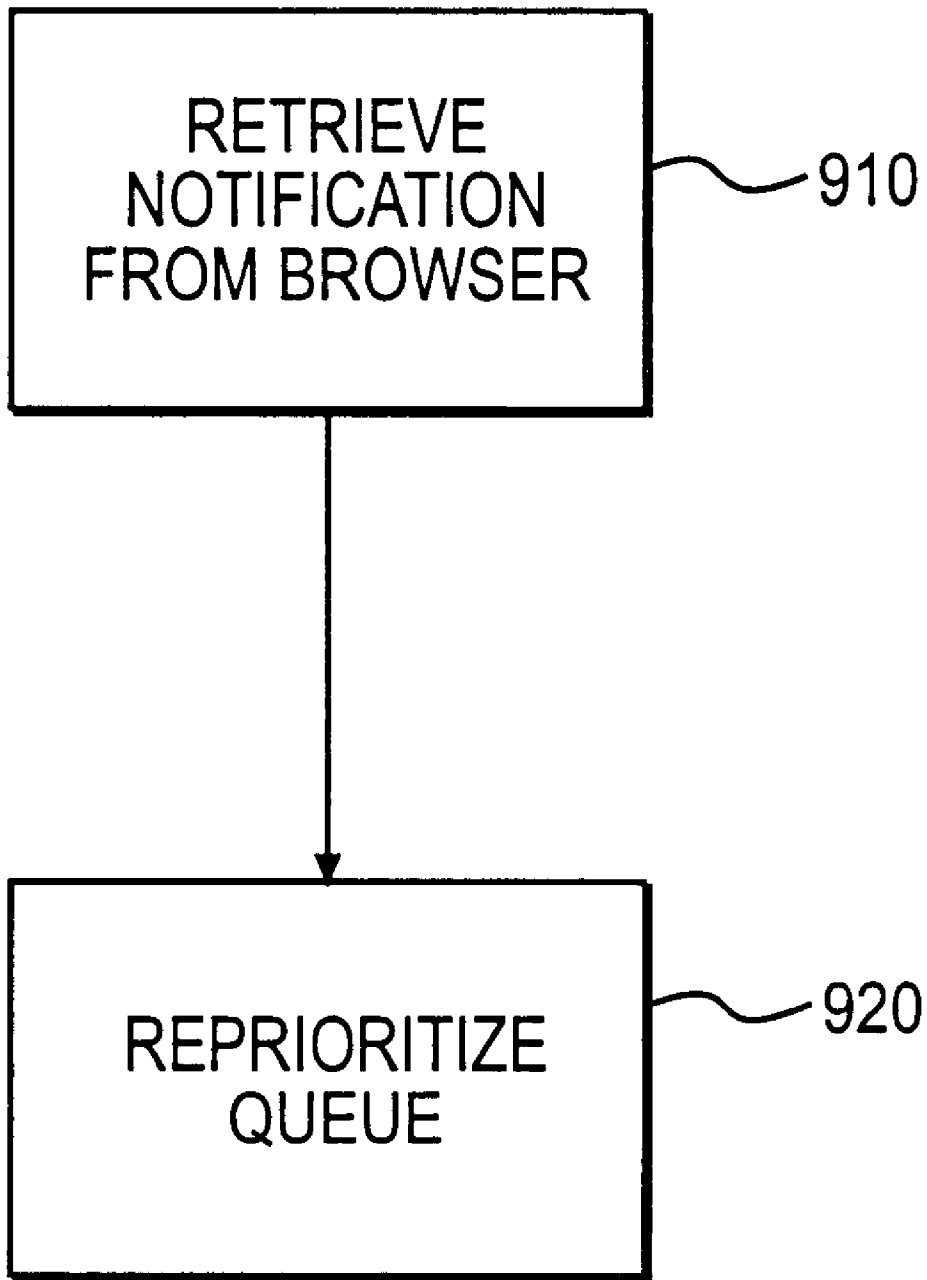
US 6,415,319 B1



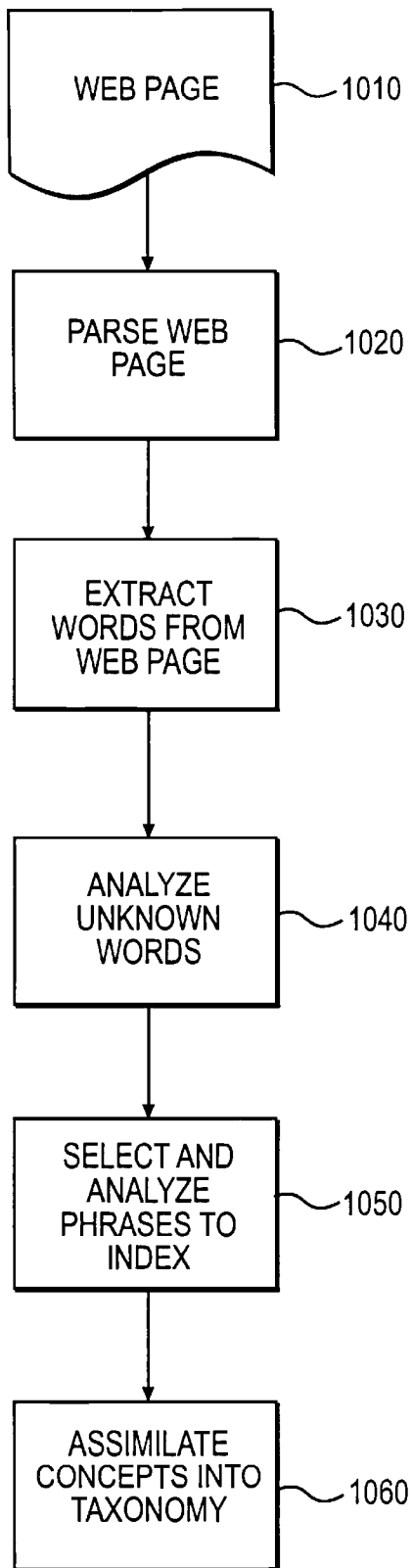
**FIG. 7**



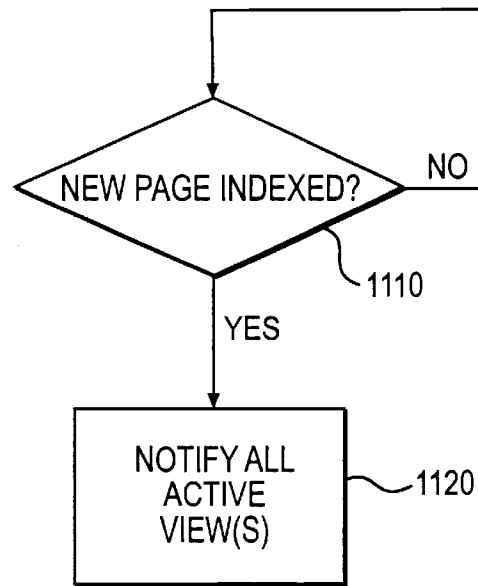
**FIG. 8**



**FIG. 9**



**FIG. 10**



**FIG. 11**

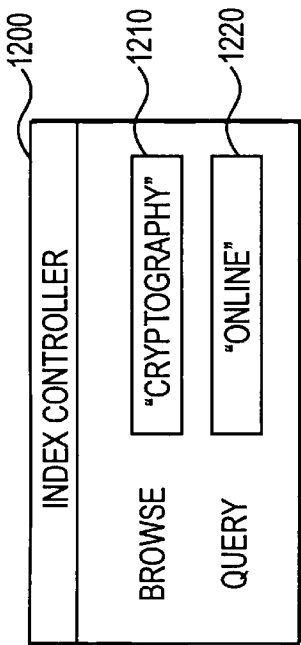


FIG. 12

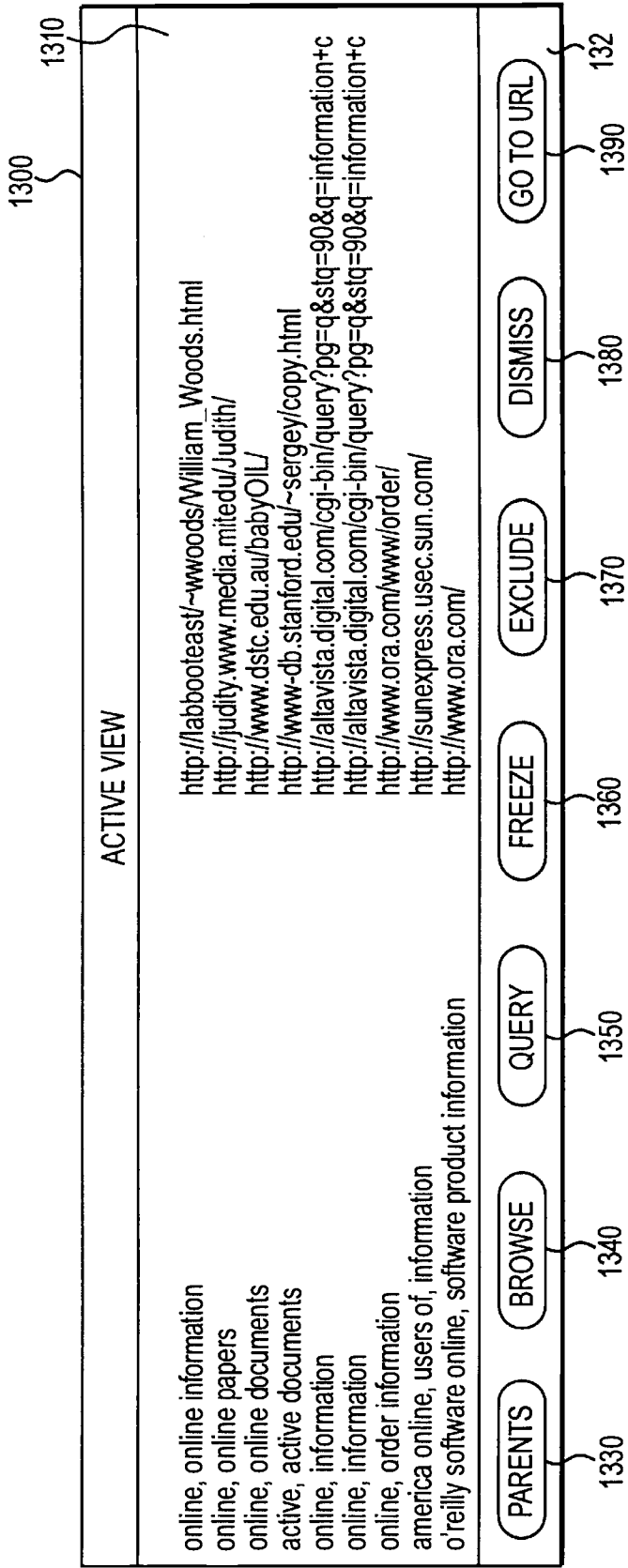


FIG. 13

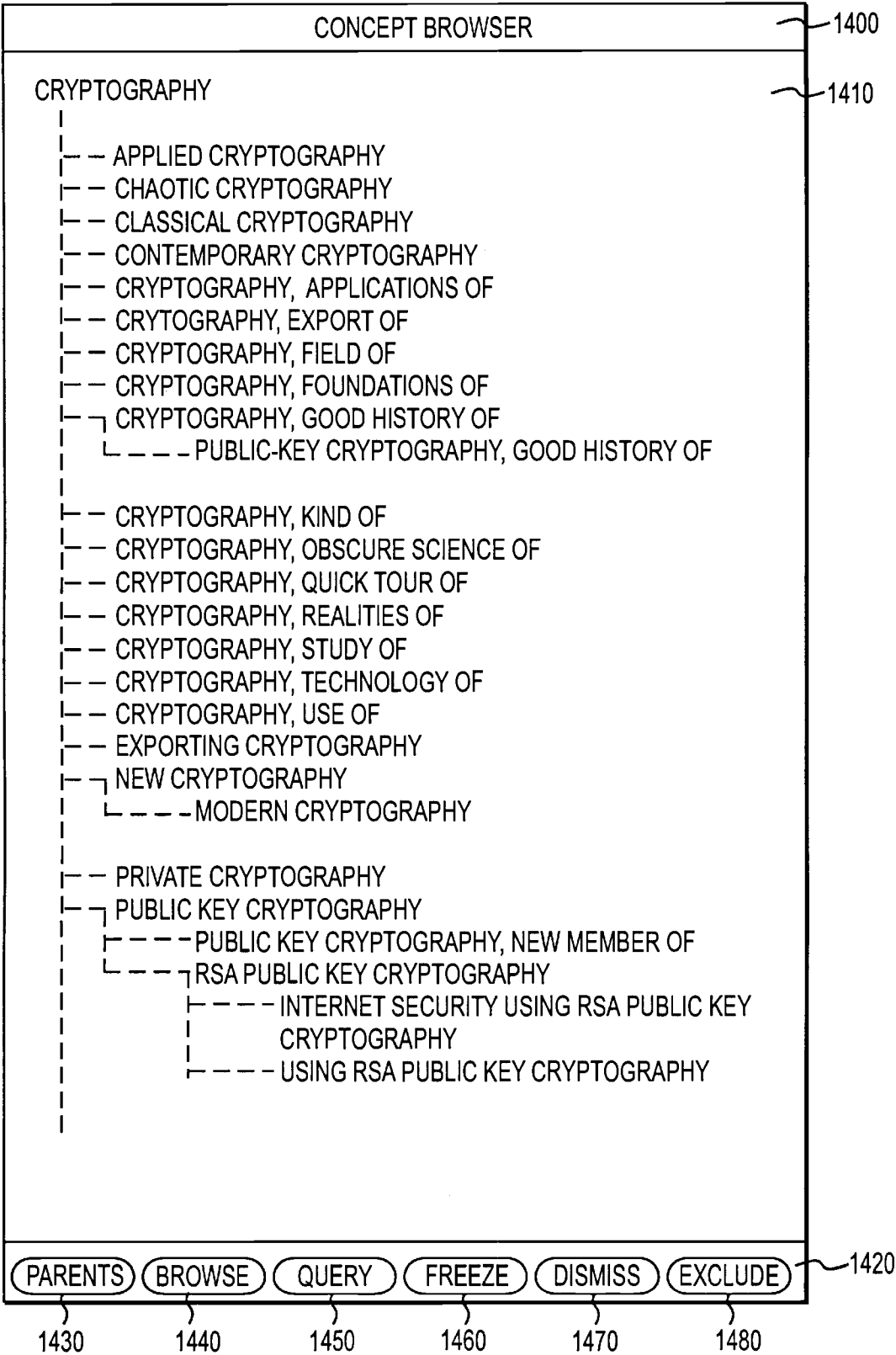


FIG. 14

US 6,415,319 B1

1

## INTELLIGENT NETWORK BROWSER USING INCREMENTAL CONCEPTUAL INDEXER

### BACKGROUND OF THE INVENTION

#### A. Field of the Invention

This invention relates generally to methods for browsing network information and, more particularly, to a method for organizing information from network documents in a conceptual index to facilitate browsing.

#### B. Description of the Related Art

The Internet, fueled by the phenomenal popularity of the World Wide Web (WWW or Web), has exhibited exponential growth over the past few years. In the case of the WWW, the ease of self-publication has helped generate an estimated 50–120 million documents.

To access all this information, users need only standard computer equipment, such as a home personal computer with a display and modem, and an Internet connection. Several types of Internet connections are available, including connections through Internet Service Providers (ISPs). To use an Internet connection from an ISP, for example, the user dials into a computer at the ISP's facility using the modem and a standard telephone line. The ISP's computer in turn provides the user with access to the Internet.

Through this Internet connection, the user accesses information on the Web using a computer program called a "Web browser," such as the Netscape Navigator™ from Netscape Communications Corporation. To accomplish this, the user gives the Web browser a Uniform Resource Locator (URL) for an object on the Internet, for example, a document containing information of interest. The document is referred to as a "Web page," and the information contained in the Web page is called "content." Web pages often refer to other Web pages using "hypertext link" or "hyperlinks" that include words or phrases representing the other pages in a form that gives the browser a URL for the corresponding Web page when a user selects a hyperlink. Hyperlinks are made possible by building Web pages using the Hypertext Markup Language (HTML).

The URL identifies a specific computer on the Internet, called a "Web Server," and, more particularly, the location of a Web page located on the Web Server. The Web browser retrieves the Web page and displays it for the user.

The virtually instantaneous and cost-free publication inherent in the WWW leads to problems with information overload. Search engines help users locate specific information on the Web; however, there is time typically only for keyword searches. As a result, one keyword search engine, Alta Vista™ from Digital Equipment Corporation, returns nearly 90,000 hits or URLs for a search for the word "zoology." Thus, the user must review the long list of URLs and access many of the corresponding Web pages to find those that contain sought-after information. This demonstrates the relative lack of utility associated with using keyword search engines available on the Internet.

Researchers are, however, experimenting with intelligent agents to facilitate browsing by "learning" the user's interests based on prior sessions surfing the Web. Two better known research prototypes include WebWatcher and Letizia.

WebWatcher is a server-based interface agent that resides between the user and the Web. Any user running a browser can enter the system simply by typing a topic of interest in WebWatcher's FrontDoor page. WebWatcher replaces the current page with a modified page that embeds WebWatcher

2

command menus and enables WebWatcher to follow the user browsing the Web; and presents the user with a highlighted listing of recommended hyperlinks. Because WebWatcher is a server-based system it logs data from thousands of users to "train" itself and refine its search knowledge. If a user signals that a particular search was successful, WebWatcher annotates each explored hyperlink with user keywords, adding to the knowledge base from previous sessions. WebWatcher uses information retrieval techniques based on the frequency of weighted terms and documents for all hyperlinks on a page, as well as user statistics associated with those links.

Letizia is a client-side personal agent and thus resides on the computer running the user's browser, as opposed to on a separate server. Letizia collects information about the user's browsing habits and tries to anticipate additional items of interest. Making inferences about user interests and using various heuristics, Letizia conducts a resource-limited search of the Web during idle times looking for promising links to suggest when prompted.

While both prototypes try to anticipate a user's interest in accessing certain information, neither addresses the problem of organizing available information on the Web to facilitate browsing. There is therefore a need for a system that organizes or indexes available network information in a structure that permits users to pinpoint the location of information likely to be of interest.

### SUMMARY OF THE INVENTION

Accordingly, systems and methods consistent with the present invention substantially obviate one or more of the problems due to limitations, shortcomings, and disadvantages of the related art by incrementally indexing conceptual information in network documents, and integrating the information in a manner usable by the user in a browsing session.

Consistent with the present invention, a method for accessing information from a network comprises the steps, performed by a processor, of: receiving a document from the network containing content; extracting conceptual information from the content of the document; analyzing the extracted conceptual information semantically; and assembling an index of the extracted conceptual information that reflects relations based on semantic data in a stored lexicon.

Both the foregoing general description and the following detailed description are exemplary and explanatory only, and merely provide further explanation of the claimed invention.

### BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate systems and methods consistent with the invention and, together with the description, explain the advantages and principles of the invention. In the drawings,

FIG. 1 is a block diagram of the software modules of a browse guide system consistent with the present invention;

FIG. 2 is a block diagram of the information flow of the browse guide system consistent with the present invention;

FIGS. 3 and 4 are flow charts of the steps performed by a BG proxy module of the browse guide system consistent with the present invention;

FIGS. 5 to 9 are flow charts of the steps performed by an index controller module of the browse guide system consistent with the present invention;

FIGS. 10 and 11 are flow charts of the steps performed by the index server module of the browse guide system consistent with the present invention;

FIG. 12 is an illustration the user interface for the index controller module of the browse guide system consistent with the present invention;

FIG. 14 is an illustration an example of an active view for the browse guide system consistent with the present invention; and

FIG. 13 is an illustration an example of a concept browser for the browse guide system consistent with the present invention.

DETAILED DESCRIPTION

Reference will now be made in detail to a system and method consistent with the present invention. Wherever possible, the same reference numbers will be used throughout the drawings and the following description to refer to the same or like parts.

Overview

Systems consistent with the present invention assist users browsing the Web by constructing a dynamic conceptual index of documents visited by the browser and documents from the immediate neighborhood of those documents, such as those connected by a hyperlink. The conceptual index is a hierarchically organized taxonomy of word and phrase concepts found in the indexed material along with corresponding locations of those concepts in the documents. Using tools to query and browse the incrementally-built conceptual index, users can access the documents at the specific location corresponding to a selected concept.

The evolving index provides two important functions: (1) an automatically assembled conceptual logbook of the user's path through the Web, and (2) a facility for conceptual "peripheral vision" that displays concepts in documents one step ahead of the browser while navigating the Web.

Conceptual Indexing

Conceptual indexing involves techniques for automatically organizing all of the words and phrases of material into a conceptual taxonomy that explicitly links each concept to its most specific generalizations. The taxonomy is a graph structure that orders concepts by generality using ISA ("is a") links. For example, the following taxonomy represents the relationship between the concepts "computer" and "laptop":

computer  
|—laptop.

In this representation, the "computer" concept is a more general form of the "laptop" concept. Thus, the "computer" concept is depicted as a parent of the "laptop" concept in the graph structure. The taxonomy can be used alone to organize information for browsing, or it can be used as an adjunct to search and retrieval techniques to construct better queries.

Conceptual indexing of text preferably involves four steps: (1) heuristic identification of phrases in the text, (2) mapping these phrases into internal conceptual structures, (3) classifying the structures into a taxonomy, and (4) linking the concept to the location of the phrase in the text. As concepts are assimilated into the conceptual taxonomy during indexing, a broad coverage English lexicon is consulted to determine semantic relationships to other concepts based on recorded knowledge about the meanings of words. If any of the words of an indexed phrase do not yet have conceptual counterparts in the evolving taxonomy, they are assimilated into the taxonomy using information from the lexicon.

For example, if the phrase "graphic workstation" is encountered when indexing a document, the lexicon is examined for the word "workstation" to learn that it is a kind of "computer," and thus assimilate the relation "workstation" ISA "computer" into the taxonomy. The process may recurse on "computer" to uncover more general relationships, all of which are added to the taxonomy. Thus, the phrase "graphic workstation" builds the following taxonomy fragment:

computer  
|—workstation  
|—graphic workstation

This example presents a portion of the taxonomy tree structure, with more specific concepts indented under their more general parents. The taxonomy does not contain all of the information from the lexicon, but only the information for words and concepts extracted from the indexed text or from other phrases assimilated into the taxonomy.

After indexing a collection of text, the taxonomy recorded for the concept "computer" might look like this:

```
computer
|-- new computer
|   |-- recent toshiba laptop
|-- toshiba computer
|   |-- recent toshiba laptop
|-- workstation
|   |-- graphic workstation
|-- server
|   |-- web server
|       |-- WWW server
|   |-- sun's new netra-j server
|-- laptop
|   |-- recent toshiba laptop
```

There are three types of relationships in the taxonomy: (1) subsumption relationships, (2) structural relationships, and (3) combination of subsumption and structural relationships. The subsumption relationships come from the lexicon. For example, the lexicon provides the following subsumption relationships in the taxonomy:

computer  
|—workstation  
|—server  
|—laptop and  
new  
|—recent.

This means that the lexicon provides the framework for building in the taxonomy structure the relationships between these concepts.

Structural relationships are derived from the phrases in the text being indexed, such as:

workstation  
|—graphic workstation.

If the lexicon does not have information required to assimilate the words into the taxonomy, the words are still assimilated into the taxonomy in accordance with structural relationships from the text.

The following is an example of a combination relationship in the above taxonomy:

new computer  
|—recent toshiba laptop.



US 6,415,319 B1

5

In this combination, the relationships between the words “new” and “recent” and the words “computer” and “laptop” are subsumption relationships from the lexicon. Using these subsumption relationships and the structural relationship from the indexed text that indicates the word “toshiba” modifies the word “laptop,” the taxonomy builds a relationship between the phrases “new computer” and “recent toshiba laptop,” as illustrated above.

The concept of a taxonomy is closely analogous to the organization of books in a library. In general, books on the same topic are located on the same shelf and in close proximity. Similarly, the taxonomy places like concepts in close proximity so that the concept “laptop” is close to “recent toshiba laptop,” the concept “server” is close to “WWW server,” and so forth.

The taxonomy aids in formulating queries. In querying the index, terms are treated as concepts and are expanded by their specific children in the taxonomy. In this way, a query for “fast computer” will be expanded to a query for “fast computer” and “fast graphic workstation” because “graphic workstation” is a more specific kind of “computer,” according to the above taxonomy.

#### System Architecture

FIG. 1 illustrates the components of a browse guide (BG) system 100 consistent with the present invention. BG 100 assists users browsing the Web by constructing a dynamic conceptual index of documents visited by the user using a browser and documents from the immediate neighborhood of those documents. BG 100 includes software modules written in the JAVA programming language. BG 100 is thus platform-independent and can run on any conventional computer, such as a personal computer with a Pentium microprocessor manufactured by Intel Corp. running the Microsoft Windows 95 operating system.

The computer is preferably equipped with hardware, such as a modem, for connecting to the Internet 160, which is depicted in FIG. 1 as the cloud surrounding Web pages 180. This is intended to show that the Web pages 180 constitute documents on the Internet 160 that are accessible to computers connected to the Internet 160.

BG 100 includes BG proxy 110, index controller 120, index server 130, index 140, and lexical database 150. BG proxy 110 performs two general functions: (1) monitoring a user’s activity browsing the Web, and (2) accessing information from Web, including documents visited by the user’s browser (not shown) and documents from the immediate neighborhood of those documents.

Index controller 120 controls the operations of BG 100 by, in part, maintaining a queue identifying the information and documents to be indexed in the conceptual taxonomy, and index server 130 performs the indexing functions, including incrementally building index 140 using lexical database 150. Lexical database 150 contains the English language lexicon used in building the conceptual taxonomy, i.e., index 140, in accordance with the methodology described above.

#### System Operation

FIG. 2 is an information flow diagram that explains the operation of BG 100. In FIG. 2, there are a number of lines connecting the modules that are labeled “URL<sub>N</sub>.” The “URL” in this label represents any URL for a site on the Internet and the subscript “N” indicates that multiple URLs may pass between the modules. There are also a number of lines labeled “WEB PAGE<sub>N</sub>.” The “WEB PAGE” in this label represents any Web page, document, file, etc. available on the Internet, and the subscripted “N” indicates that multiple Web pages may pass between the modules. The

6

single quote (‘) next to the “N” on a number of the lines with the WEB PAGE<sub>N</sub> label indicates that the page has been modified from its original state.

When a user 205 enters a URL into browser 210, which may be a conventional Web browser such as Microsoft Explorer®, he requests to retrieve a Web page from the Internet 160. BG proxy 110 intercepts the URL before providing it to network server 220 on the Internet 160, to retrieve the requested Web page. Network server 220 then provides the Web page identified by the URL to the BG proxy 110, which in turn provides the page to browser 210 for display. However, the retrieved Web page is modified by BG proxy 110. In part, this modification enables index controller 120 to connect to browser 210 and monitor its activity.

BG proxy 110 also provides the retrieved Web page to index controller 120, and Index controller 120 passes the Web page to index server 130 for indexing. The indexing process involves parsing the retrieved Web page, and assimilating the concepts in the Web page into taxonomy index 140 using lexical database 150. In parsing the retrieved Web page, index server 130 also determines the presence of any Web pages referenced in the retrieved Web page by identifying the hyperlinks in that page. If there are any referenced Web pages, index server 130 provides the URLs for those pages to index controller 120.

Index controller 120 maintains a priority queue of URLs for Web pages to be indexed by index server 130. The queue includes the URLs referenced in pages previously indexed. Preferably, the priority is set according to the user’s activity, although other priorities are possible. Thus, if the user selects one of the URLs in the queue or another URL not in the queue, index controller 120 causes the retrieval of the corresponding Web page and provides that page to index server 130 for assimilation of the page’s concepts into taxonomy index 140 ahead of Web pages for any other URLs in the queue. Additionally, while the queue is not empty, index controller 120 provides each URL in the queue to BG proxy 110 to retrieve the corresponding Web page, which is in turn passed through index controller 120 to index server 130 for assimilation of the concepts in that Web page into index 140.

User 205 can also query index 140. By inputting a query word or phrase, user 205 instructs index controller 120 to generate and display an active view 230 that includes concepts and corresponding URLs from index 140. When index controller 120 receives a request for an active view, it passes the request to index server 130 to access index 140 for the concepts containing the search terms.

BG 100 builds connections between browser 210 and index controller 120, index controller 120 and each active view 230, and index server 130 and each active view 230. These three connections are labeled (1), (2), and (3) in FIG. 2. A key table at the bottom of FIG. 2 explains the information flow along these connections. The first connection (1) between browser 210 and index controller 120 enables index controller to monitor user 205 activity for managing the queue priority.

The second connection (2) between index controller 120 and active view 230 exists as a result of index controller 120 creating active view 230 in response to a request from user 205 and links active view 230 to browser 210 via the first connection (1). This link enables the user to select a concept from active view 230 for display of a corresponding Web page by browser 210.

The third connection (3) between index server 130 and active view 230 exists as a result of index controller 120

US 6,415,319 B1

7

creating active view **230** in response to a request from user **205** and links active view **230** to index server **130**. This link enables index server **130** to update or refresh the active view **230** when additional information is assimilated into index **140** and active view **230** includes a fragment of index **140** modified by this assimilation. In this fashion, information is dynamically organized in real-time during a browsing session.

#### Process

##### BG Proxy Process

FIGS. **3** and **4** are flow diagrams illustrating the steps of two processes performed by BG proxy **110**. The first process relates to the operation intercepting URLs from browser **210** for indexing of the corresponding Web pages, and the second concerns the functions associated with the queue processing operation of index controller **120**.

When user **205** enters a URL into browser **210**, BG proxy **110** intercepts the input URL from browser **210** (step **300**). BG proxy **110** then sends the intercepted URL to network server **220** on the Internet to retrieve the corresponding Web page (step **310**). When the Web page is retrieved (step **320**), BG proxy **110** embeds a "plug in" into the Web page (step **330**). BG proxy **110** provides the modified Web page to browser **210** for display (step **340**) and to index controller **120** for further processing by index server **130** (step **350**). The embedded "plug-in" is a computer program written in, for example, the C++ programming language, and enables index controller **120** to connect to browser **210** and to monitor activity of browser **210**. This in turn enables index controller **120** to prioritize the retrieval of Web pages corresponding to entries in the URL queue.

When URL queue of the index controller **120** is not empty, index controller **120** provides each URL in the queue to BG proxy **110**. After BG proxy receives a URL from index controller **120** (step **400**), BG proxy **110** transmits the URL to the appropriate network server in the Internet to retrieve the Web page corresponding to the URL (step **410**). When BG proxy **110** receives the Web page (step **420**), BG proxy **110** embeds the plug-in for index controller **120** (step **430**), and then provides the modified Web page to index controller **120** (step **440**), which will in turn provide the page to index server **130** for assimilation of the concepts in the page into index **140**.

##### Index Controller Process

FIGS. **5** through **9** are flow charts illustrating the steps of five processes performed by index controller **120**. The first process of index controller **120** (see FIG. **5**) is a pass-through function. When index controller **120** receives a modified Web page from BG proxy **110** (step **510**), index controller provides the modified Web page to index server **130** for assimilation (step **520**).

The second process of index controller **120** (see FIG. **6**) concerns the URL queue. As described above, index server **130** identifies any URLs in Web pages being indexed. Index controller **120** receives a set of URLs from index server **130** for each Web page that has been indexed (step **610**). The set may be empty if the indexed Web page contains no hyperlinks referencing other URLs and Web pages. Each URL in the set is added to the index controller's **120** queue for Web page retrieval and processing in accordance with a priority (step **620**). For example, index controller **120** sets a high priority for the set of URLs corresponding to the Web page currently displayed by browser **210**. In this manner, browse guide **100** assimilates the pages neighboring the currently displayed page before working on pages for other URLs in the queue.

The next process of index controller **120** (see FIG. **7**) also concerns queue processing. The process of FIG. **6** relates to

8

adding items to the queue, and the process of FIG. **7** concerns taking URLs off the queue. First, index controller **120** is constantly monitoring its queue to determine whether there are any URL entries (step **710**). If there are no URLs in the queue, then index controller **120** remains in the monitoring state (step **710**). If the queue is not empty (step **710**), then index controller **120** selects the next URL in the queue with the highest priority and fetches that URL from the queue (steps **720** and **730**). Index controller **120** then provides the URL to BG proxy **110** (step **740**), which, as explained above with reference to FIG. **4**, retrieves the corresponding Web page from the Internet.

The fourth index controller **120** process (see FIG. **8**) concerns the second connection (2) (see FIG. **2**) between active view **230** and index controller **120**. Browse guide **100** includes an interface, such as a display with dialogue boxes, for users to browse and query index **140**. An exemplary interface **1200** is shown in FIG. **12** and contains a display with two boxes **1210** and **1220**. In the first box **1210**, users can input a concept to view a fragment of index **140**. As shown, box **1210** includes the word "cryptography". BG **100** would in turn display in a "concept browser" a portion of index **140** with the concept "cryptography" as the most general concept (i.e., at the root) with the more specific concepts branched below. An exemplary concept browser display **1400** for the "cryptography" concept from an exemplary index is shown in FIG. **14**.

Display **1400** includes two part: the first part **1410** is the requested taxonomy fragment, and the second part **1420** is for buttons **1430** to **1480** that the user can select to initiate various operations by pointing a mouse icon on a button and clicking on the mouse button. Parents button **1430** is used to display the parents of a select, highlighted concept in area **1410**. Browse button **1440** is used to browse through index **140**. Query button **1450** is used to allow the user to build an active view from a selected, highlighted concept. Freeze button **1460** is used to instruct the system not to modify area **1410**, regardless of whether index **140** assimilates additional Web pages. Dismiss button **1470** is used to instruct browse guide **100** to close the display, and exclude button **1480** is used to eliminate a highlighted concept from the area **1410**.

The second box **1220** in interface **1200** is for the user to select and input a query term for BG **100** to create an active view (i.e., query index **140**). Although FIG. **2** shows only one active view, BG **100** preferably supports multiple active views as well as multiple concept browsers so the user can work with many of both view. As shown in FIG. **12**, the word "online" in box **1220** is for an active view, and the exemplary active view for the "online" query is shown in FIG. **13**.

Display **1300** includes two part: the first part **1310** is the results of a query on index **140** with corresponding URLs, and the second part **1320** is for buttons **1330** to **1390** that the user can select to initiate various operations by pointing a mouse icon on a button and clicking on the mouse button. The query results are ordered or ranked based on a quality of match between the query phrase and text in Web pages using the concepts in index **140**. This process is described in U.S. patent application Ser. No. 08/499,268, for "Method and Apparatus for Generating Query Responses in a Computer-Based Document Retrieved System," filed Jul. 7, 1995, which is incorporated herein. Buttons **1330** to **1380** correspond to buttons **1430** to **1480**, respectively, and perform the same functions as those described above with reference to buttons **1430** to **1480**. When selected by the user, the new "GO TO URL" button **1390** instructs index controller **120** to provide the URL for a highlighted word or

US 6,415,319 B1

9

phrase to browser **210** to in turn access the identified server and to retrieve and display the corresponding Web page.

Returning to FIG. **8**, index controller **120** receives from the user a request to query index **140** in the form of a request for an active view (step **810**). When index controller **120** receives such a query, it sets up and displays the requested active view with the results of the query of index **140** (step **820**), and connects the active view to index server **130** (step **830**). This connection enables index server **130** to update the displayed active view as index **140** changes, assimilating concepts from additional pages.

The fifth process of index controller **120** (see FIG. **9**) concerns the connection between index controller **120** and browser **210** (see (1) in FIG. **2**). Index controller monitors browser **210** activity, retrieving notifications from browser **210** as the displayed Web pages selected by the user are displayed (step **910**). In response to such notification, index controller **120** reprioritizes the URLs in the queue so that browse guide **100** assimilates concepts of the corresponding Web pages in a priority that closely matches the user's interest, as demonstrated by the Web page currently displayed by browser **210**.

#### Index Server Process

Index server **130** maintains taxonomy index **140** by assimilating concepts from new Web pages into the taxonomy using information from the Web pages and from the relations in lexical database **150**. Flow charts of the steps of the two processes performed by index server **130** are shown in FIGS. **10** and **11**. In FIG. **10**, index server **130** receives as input a Web page (step **1010**). While this description focuses on indexing Web pages from the Internet, information from other documents, such as e-mail messages, text files, and databases, may also be assimilated into the taxonomy.

Index server **130** parses the Web page to identify text in the page for assimilation (step **1020**), and then extract words from the Web page (step **1030**). This extraction process is commonly referred to as "chunkifying" the text, a process by which a set of predetermined rules is used to determine the text in the page that correspond to words. This can be done by a conventional process that involves examining the page for spaces in between characters and punctuation.

Index server **130** then identifies the words that do not appear in lexical database **150** and analyzes those "unknown" words to determine whether they represent concepts capable of assimilation into index **140** (step **1040**). This step involves a morphological process in which index server **130** determines whether the unknown words include other known words that exist in lexical database **150**. For example, index server **130** parses the word "sparcstation" to determine that it is comprised of two words "sparc" and "station", and to infer from this morphology where to fit the unknown word "sparcstation" in the taxonomy.

After all of the words and phrases in the Web page have been identified (steps **1030** and **1040**), index server **130** tags them as different parts of speech, for example, noun, adjective, etc. (step **1050**), using known techniques, such as the extracting operation taught by Eric Brill, "Some Advances in Rule-Based Part of Speech Tagging," AAAI Conference, 1994. Index server **130** then assimilates the words and phrases extracted from the Web page into index **140** along with a reference to the location of each word and phrase in the Web page. When a user selects the phrase from an active view, browse guide **100** instructs browser **210** to display a Web page with a highlighted passage corresponding to the selected phrase.

Finally, index server **130** also performs operations in connection with updating active views. When a new page is

10

indexed (step **1110**), index server **130** updates all active views affected by changes to index **140**. For example, the active view for "online" described above is updated to reflect changes to index **140**, such as when additional Web pages that include the word "online" and related concepts are assimilated into index **140**.

#### Conclusion

To overcome the shortcomings of conventional intelligent agents, the present invention automatically organizes information retrieved during a session browsing the Internet in a conceptual index to facilitate the browsing process. Using tools to query and browse the incrementally-built conceptual index, users can access the documents at the specific location corresponding to a selected concept. Since the indexing process involves not only retrieving information from pages actually visited but also from neighboring pages, the present invention provides a "peripheral vision" that displays concepts in documents one step ahead of the browser while navigating the Web.

The foregoing description of an implementation of the invention has been presented for purposes of illustration and description. It is not exhaustive and does not limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practicing of the invention. For example, the described implementation includes software but the present invention may be implemented as a combination of hardware and software or in hardware alone. The scope of the invention is defined by the claims and their equivalents.

#### What is claimed is:

**1.** A computer-implemented method for accessing documents from a network, comprising:

providing a browser for accessing documents from the network;

registering a request for an active view corresponding to a designated term;

intercepting a request to access a first document;

determining whether the first document includes the designated term;

automatically accessing a second document, wherein the second document is referenced in the first document;

determining whether the second document includes the designated term; and

automatically updating the active view to indicate the presence of the designated term in the first document if the first determination is positive and to indicate the presence of designated term in the second document if the second determination is positive.

**2.** The method of claim **1**, wherein the step of updating the active view includes automatically displaying the active view.

**3.** The method of claim **1**, further comprising displaying the document, wherein the displayed document includes a visual indication of a portion of the document including the designated term.

**4.** A system for accessing documents from a network, comprising:

a conceptual index identifying specific terms and corresponding locations within documents where specific terms can be found;

a browser for accessing documents from a network; and

a proxy server for automatically updating the conceptual index to reflect a location within at least one new document for at least one of the specific terms when the browser accesses the new document from the network.

11

5. The system of claim 4, further comprising:  
an active view registered with the proxy server to be  
displayed when updating the conceptual index.  
6. A method for accessing information from a network  
comprising the steps, performed by a processor, of: 5  
retrieving from the network a document containing con-  
tent in response to a user's request;  
monitoring the user's behavior accessing documents from  
the network; 10  
extracting conceptual information from the content of the  
retrieved document in a manner reflecting the user's  
behavior;  
analyzing the extracted conceptual information semanti-  
cally; and 15  
assembling an index of the extracted conceptual informa-  
tion that reflects relations based on semantic data in a  
stored lexicon,  
wherein the extracting step includes: 20  
assembling a queue containing references to additional  
documents referenced in the retrieved document; and  
reordering the references to additional documents  
assembled in the queue based on the monitored  
user's behavior.  
7. A method for accessing information from a network 25  
comprising the steps, performed by a processor, of:  
retrieving from the network a document containing con-  
tent in response to a user's request;  
monitoring the user's behavior accessing documents from 30  
the network;  
extracting conceptual information from the content of the  
retrieved document in a manner reflecting the users  
behavior;

12

analyzing the extracted conceptual information semanti-  
cally; and  
assembling an index of the extracted conceptual informa-  
tion that reflects relations based on semantic data in a  
stored lexicon,  
Wherein the monitoring step comprises:  
embedding in the retrieved document a plug-in to  
enable monitoring of the user's behavior.  
8. A system for accessing documents from a network,  
comprising:  
a browser for receiving, from a user, requests to access  
documents from a network and displaying accessed  
documents;  
a network server;  
a proxy for intercepting the user's requests;  
an index controller for controlling activity of the proxy  
and monitoring activity of the browser;  
an index identifying specific terms and corresponding  
locations within documents where specific terms can be  
found;  
a lexical database; and  
an index server for indexing information and documents  
in the index based on the lexical database,  
wherein the proxy sends the intercepted requests to the  
network server, receives accessed documents from  
the network server, modifies each document by  
embedding a plug-in into the document, and pro-  
vides each modified document to the browser and the  
index controller.

\* \* \* \* \*

UNITED STATES PATENT AND TRADEMARK OFFICE  
**CERTIFICATE OF CORRECTION**

PATENT NO. : 6,415,319 B1  
DATED : July 2, 2002  
INVENTOR(S) : Jacek Ambroziak

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 11,


Line 33, "the users" should read -- the user's --; and

Column 12,

Line 6, "Wherein" should read -- wherein --.

Signed and Sealed this

Nineteenth Day of August, 2003

A handwritten signature in black ink, appearing to read "James E. Rogan", with a long horizontal stroke underneath.

JAMES E. ROGAN  
*Director of the United States Patent and Trademark Office*



(12) **United States Patent**  
**McCollum et al.**

(10) **Patent No.:** **US 6,594,691 B1**  
(45) **Date of Patent:** **Jul. 15, 2003**

(54) **METHOD AND SYSTEM FOR ADDING FUNCTION TO A WEB PAGE**

6,415,319 B1 \* 7/2002 Ambroziak ..... 709/219

OTHER PUBLICATIONS

(75) Inventors: **Charles P. McCollum**, Phoenix, AZ (US); **Andrew L. Burgess, Jr.**, Desert Hills, AZ (US)

(73) Assignee: **Surfnet Media Group, Inc.**, Tempe, AZ (US)

(\*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

“Information Bulletin: Internet Cookies” U.S. Department of Energy Computer Incident Advisory Capability, Mar. 12, 1998.\*  
“RealThings Design Guide” IBM, 1998.\*  
\* cited by examiner

(\*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

*Primary Examiner*—Zarni Maung  
*Assistant Examiner*—Gregory Clinton  
(74) *Attorney, Agent, or Firm*—Jordan M. Meschkow; Lowell W. Gresham; Charlene R. Jacobsen

(21) Appl. No.: **09/429,357**

(57) **ABSTRACT**

(22) Filed: **Oct. 28, 1999**

(51) **Int. Cl.**<sup>7</sup> ..... **G06F 15/16**

(52) **U.S. Cl.** ..... **709/218; 709/219**

(58) **Field of Search** ..... 709/219, 203, 709/224, 225, 227, 281, 310, 217

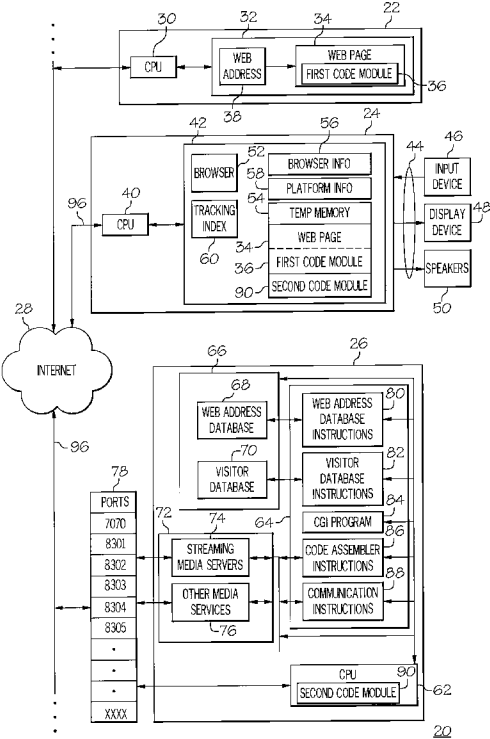
A computer network (20) includes a first processor (22) for maintaining a Web page (34) having an embedded first code module (36) and accessible through a Web address (38). A second processor (24) supports a Web browser (52) for downloading the Web page (34) and executing the first code module (36). When executed, the first code module (36) issues a first command (93) to retrieve a second code module (90) from a server system (26). The server system (26) includes a database (68) having a service response (162, 176, 186) associated with the Web address (38). A processor (62) assembles the second code module (90) having the service response (162, 176, 186). When the second code module is retrieved, the first code module (36) issues a second command (106) to initiate execution of the second code module (90) to provide added function to the Web page (34).

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,845,075 A 12/1998 Uhler et al. .... 395/200.3  
5,903,727 A 5/1999 Nielsen ..... 395/200.42  
6,009,410 A \* 12/1999 LeMole et al. .... 709/219  
6,112,240 A \* 8/2000 Pogue et al. .... 709/224  
6,128,655 A \* 10/2000 Fields et al. .... 709/225  
6,212,564 B1 \* 4/2001 Harter et al. .... 709/227  
6,317,761 B1 \* 11/2001 Landsman et al. .... 709/231  
6,327,609 B1 \* 12/2001 Ludewig et al. .... 709/203  
6,401,134 B1 \* 6/2002 Razavi et al. .... 709/310

**28 Claims, 11 Drawing Sheets**



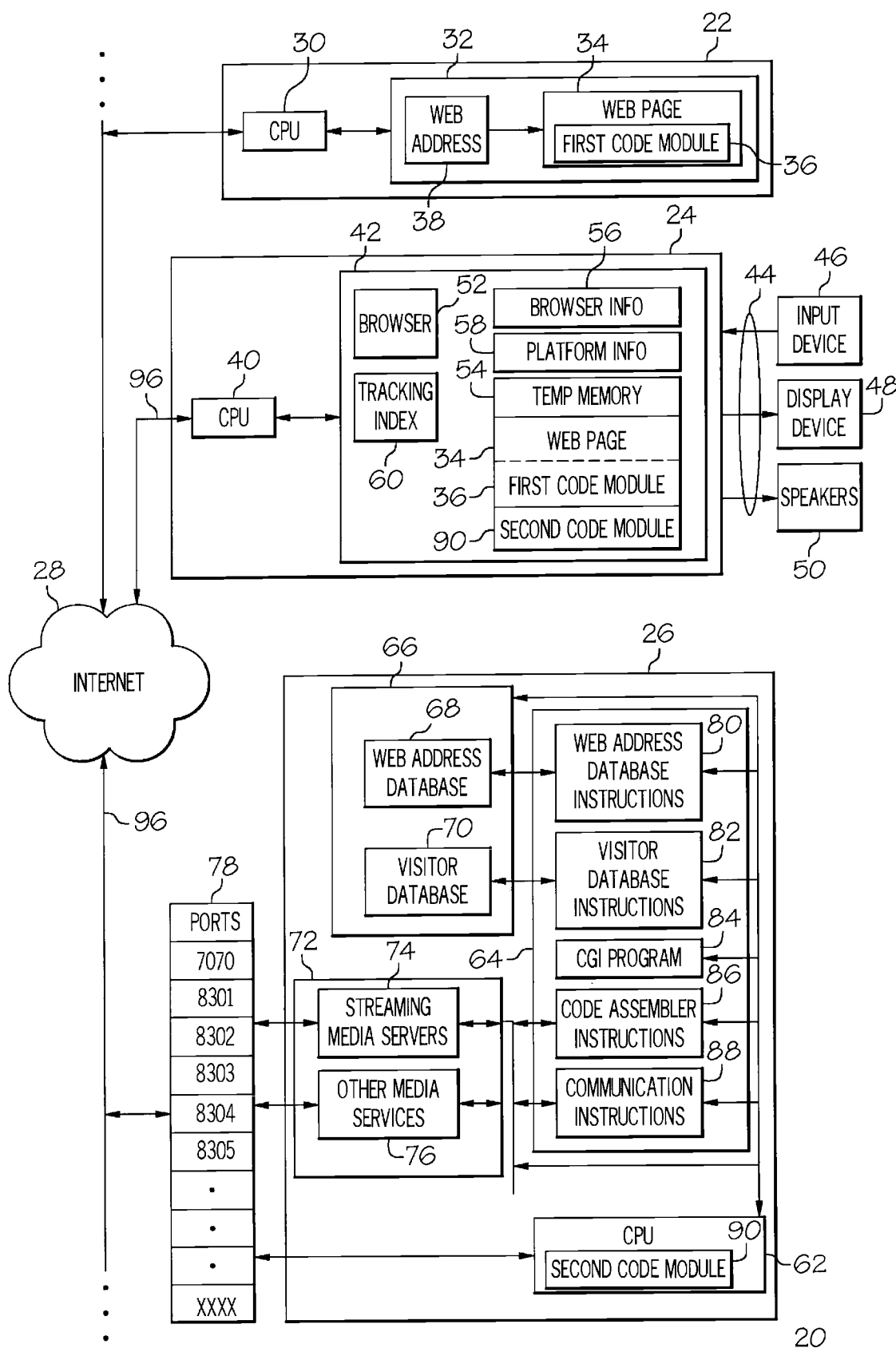


FIG. 1



LINE NO.	93 CODE
92 1	<script src= 'http://bslserver.domainname.com/ cgi-bin/bslservercall.cgi> 94
98 2	</script>
100 3	<script><!-- 102
104 4	BSLStart (); 106
108 5	//--></script> 102

36

FIG. 2

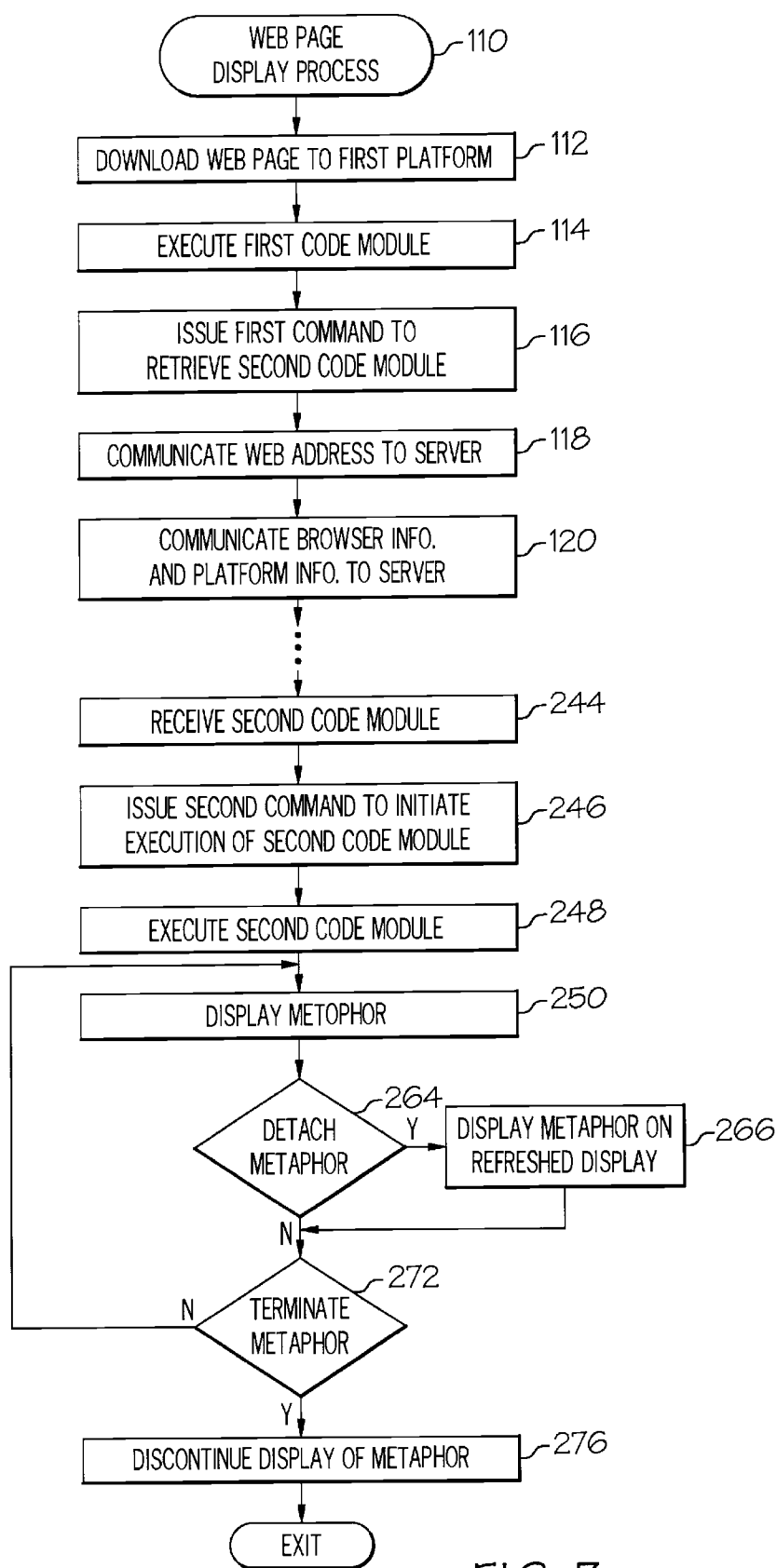


FIG. 3

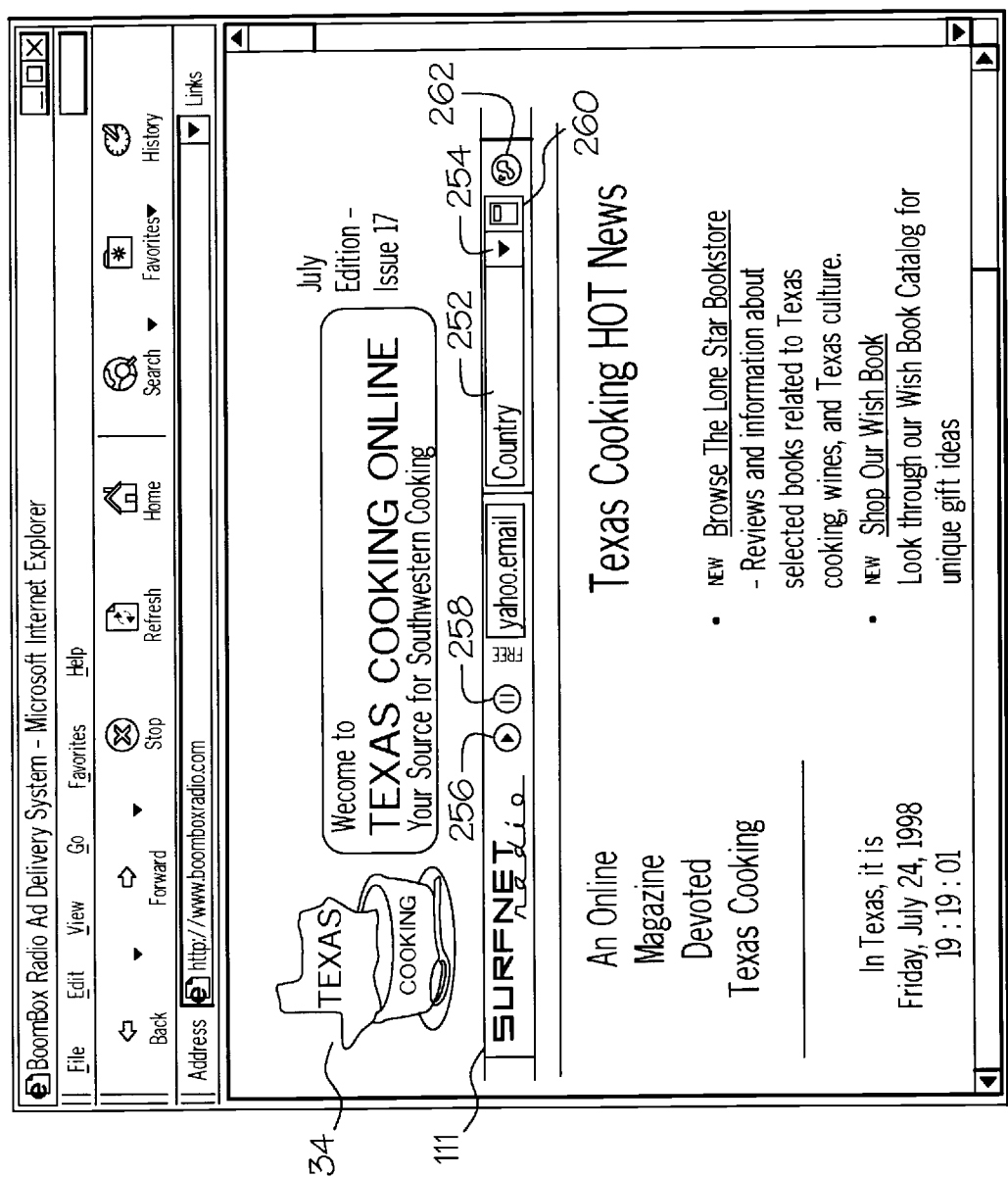


FIG. 4

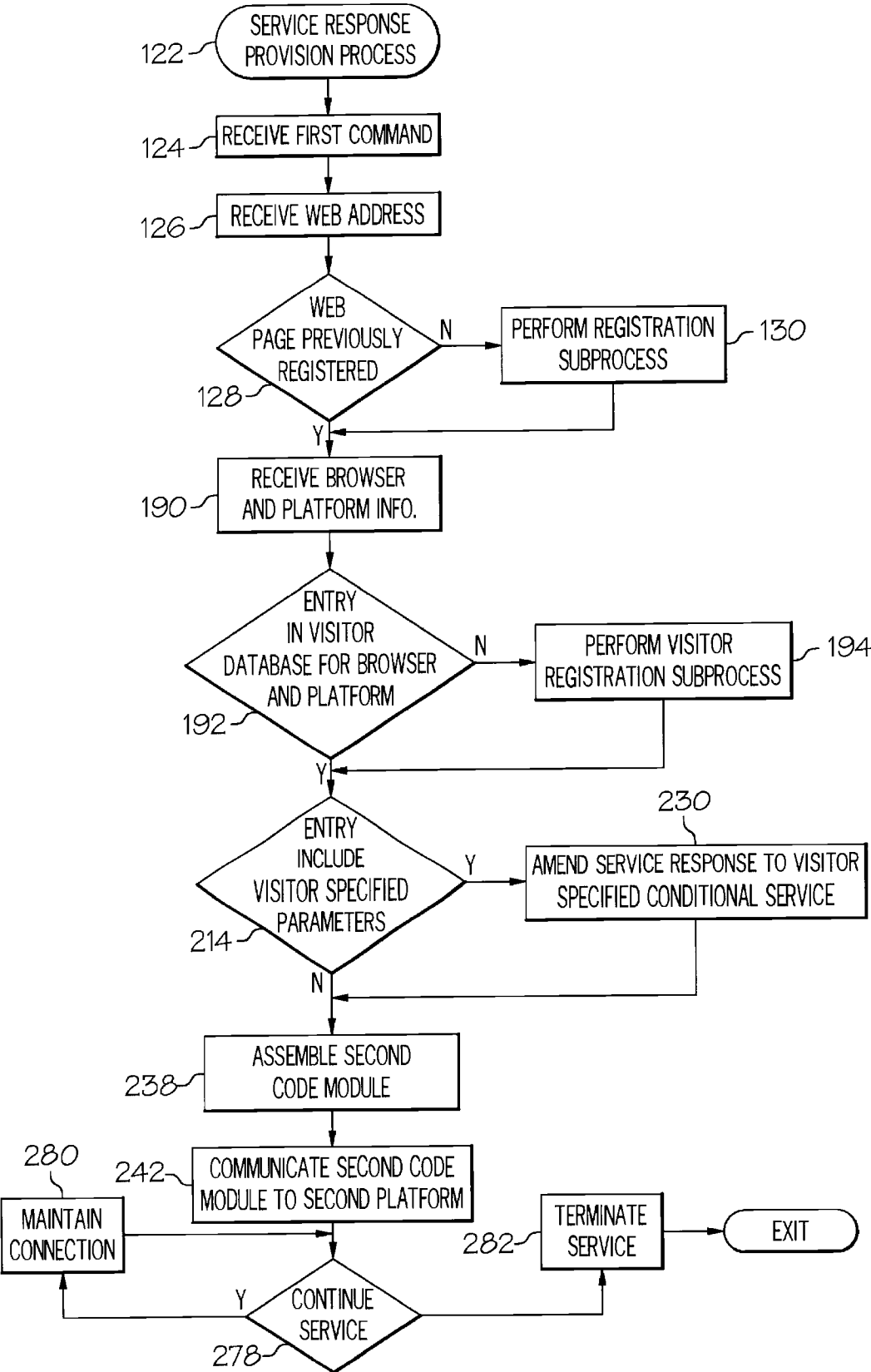


FIG. 5

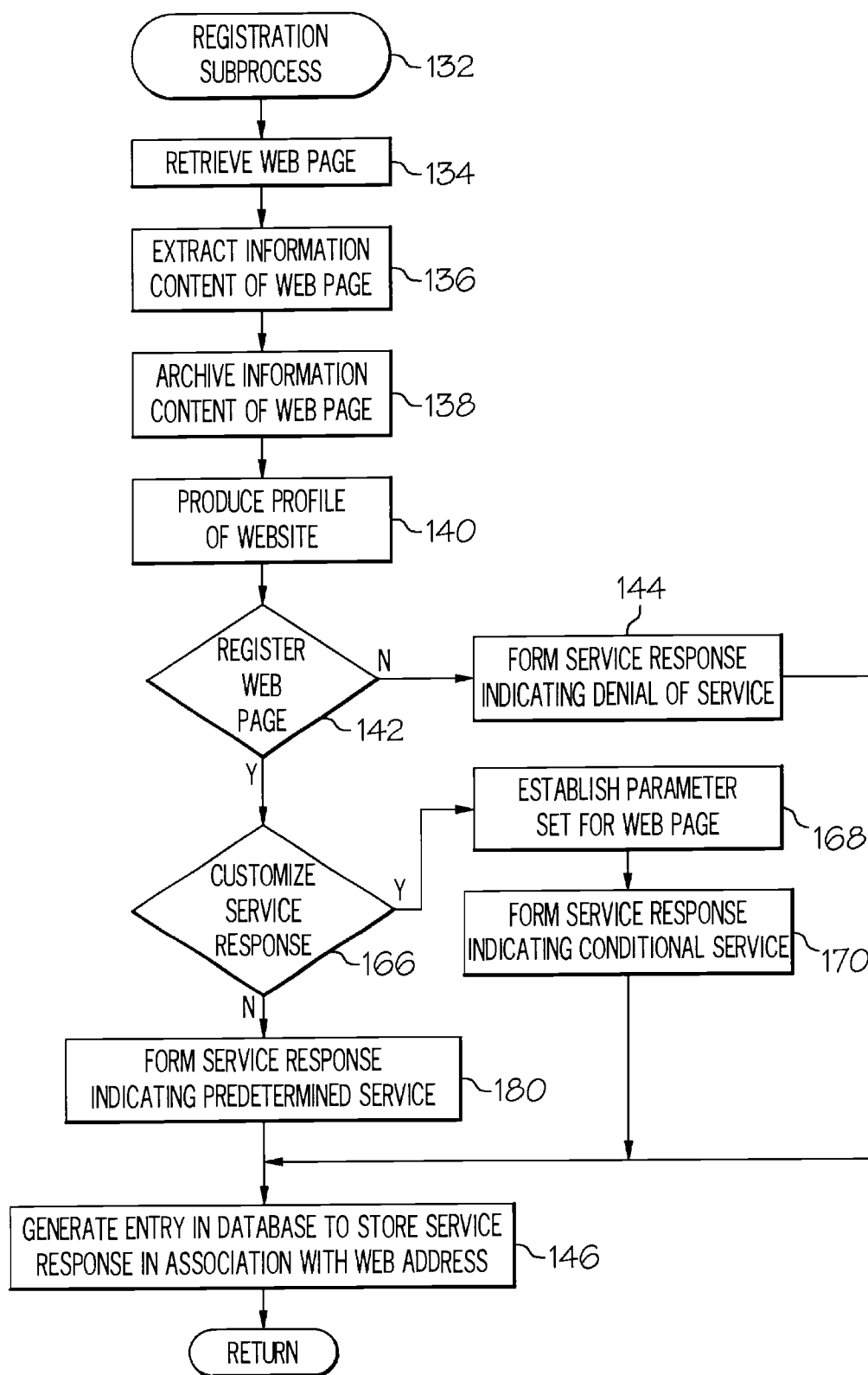


FIG. 6

150		152	154	
WEB ADDRESS FIELD		PROFILE FIELD	SERVICE RESPONSE FIELD	PARAMETER SET FIELD
158 38 172 182 38 232	URL 1 160	RECREATION/ GOLF	DENIAL OF 176 SERVICE	DENIAL CONTENT
	URL 2 174	TEXAS COOKING	CONDITIONAL 186 SERVICE	CONDITIONAL CONTENT (INCLUDING URL 5)
	URL 3 184	WEDDING	PREDETERMINED 186 SERVICE	PREDETERMINED CONTENT
	URL 4	FOOTBALL 234	PREDETERMINED SERVICE (FLAG- CONDITIONAL SERVICE FOR TRACKING INDEX 60)	PREDETERMINED CONTENT
	⋮		⋮	
	URL n		.	

FIG. 7

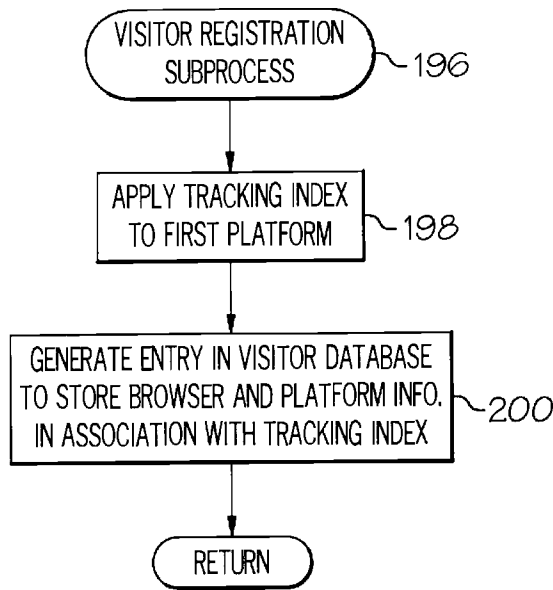


FIG. 8

	202	204	206	
	TRACKING INDEX	BROWSER ID	PLATFORM ID	VISITOR PREFERENCES
210	60	210	210	210
	SECOND PLATFORM	BROWSER INFO	PLATFORM INFO	VISITOR SPECIFIED PATAMETER SET
		56	58	

FIG. 9



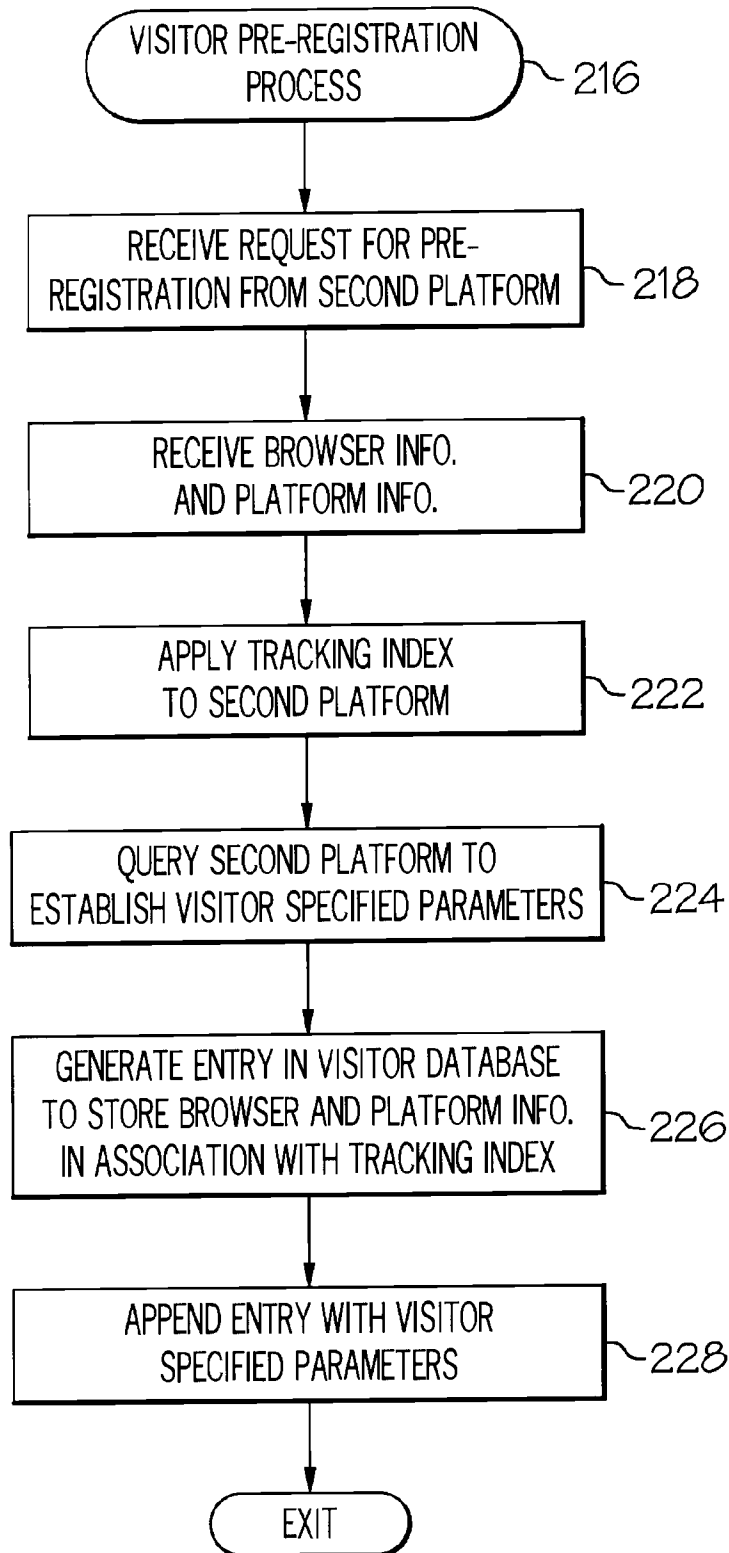


FIG. 10

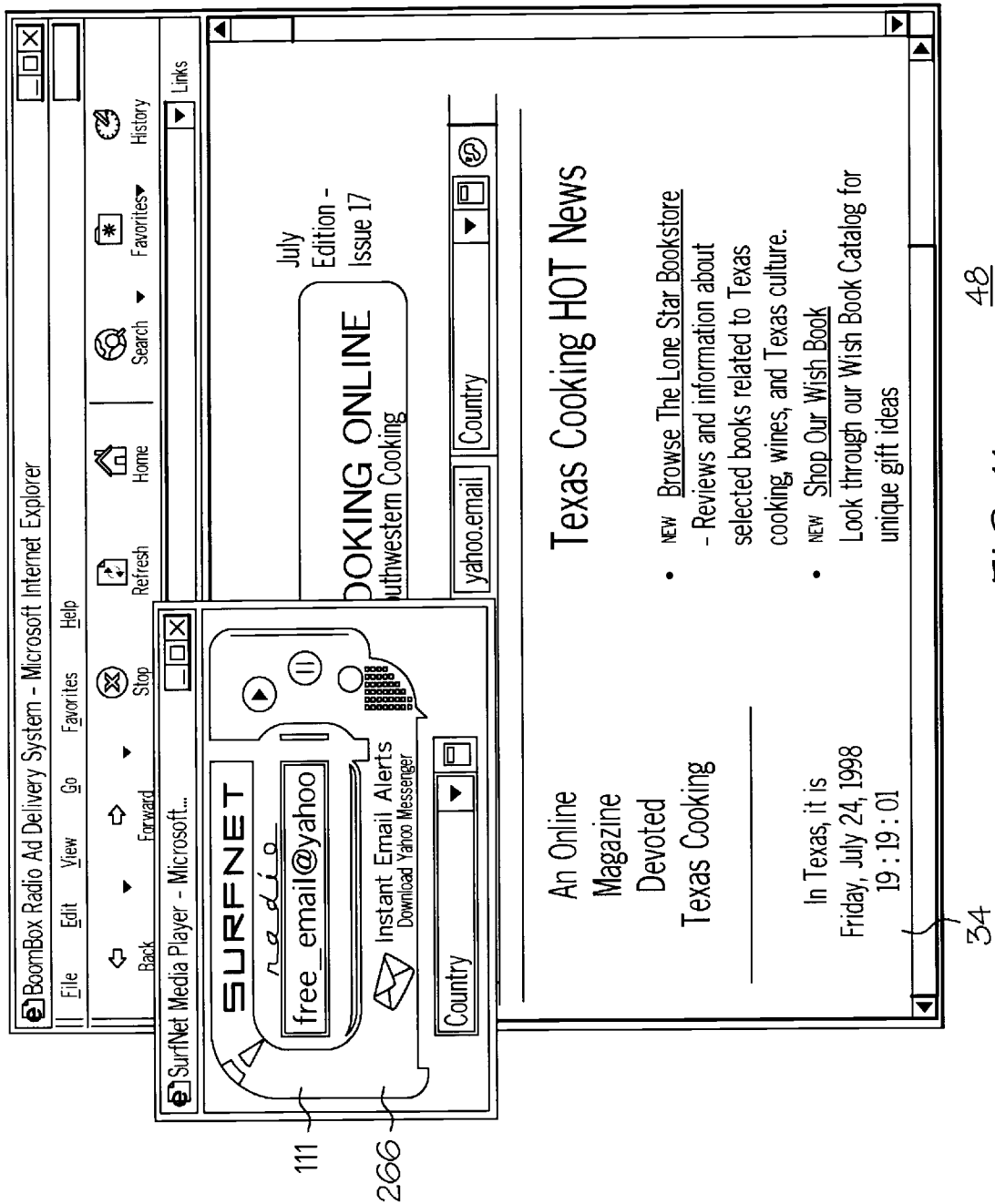


FIG. 11

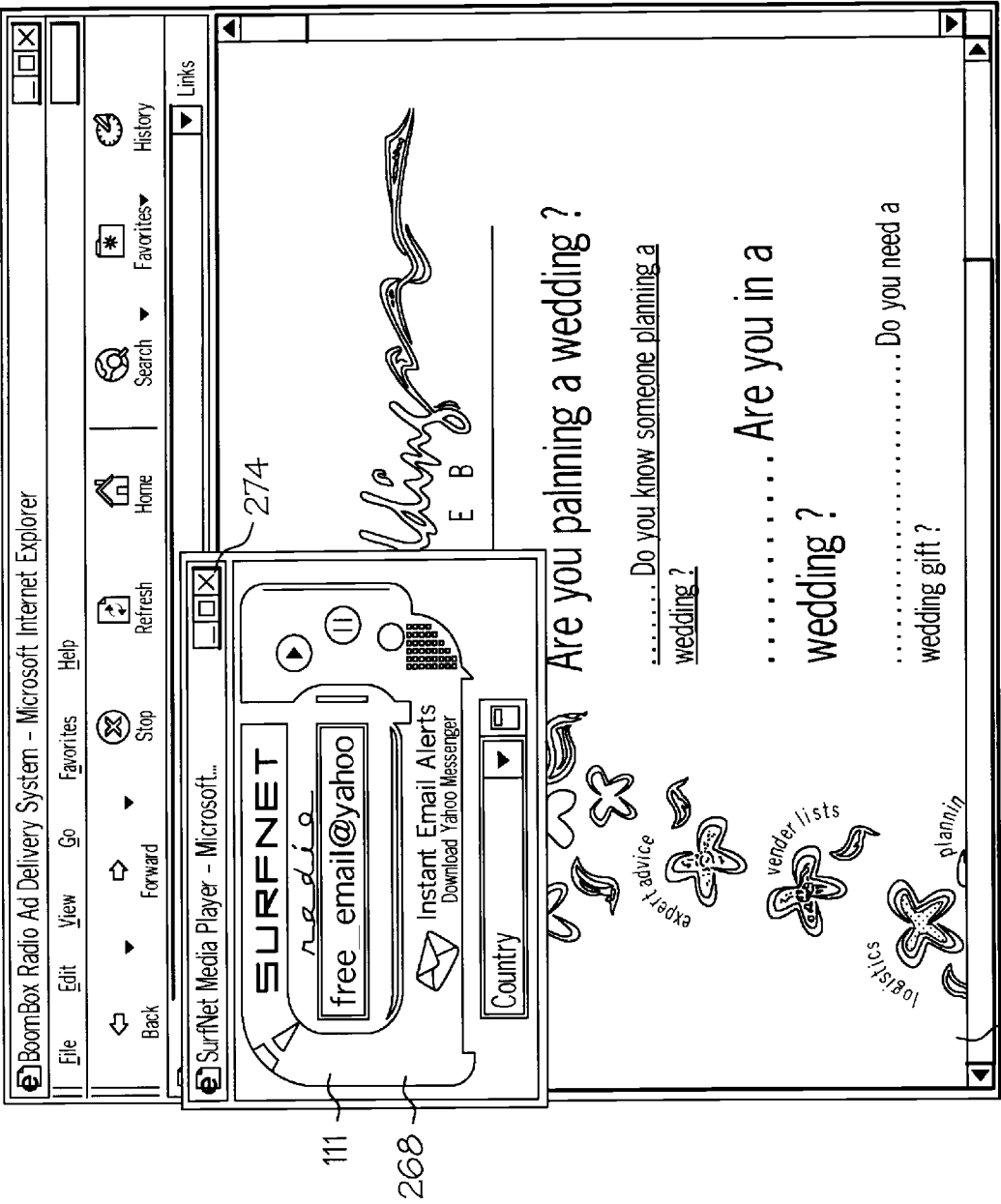


FIG. 12

US 6,594,691 B1

1

## METHOD AND SYSTEM FOR ADDING FUNCTION TO A WEB PAGE

### TECHNICAL FIELD OF THE INVENTION

The present invention relates to the field of computer networks. More specifically, the present invention relates to methods and systems for adding function to Web pages that are accessible through the Internet.

### BACKGROUND OF THE INVENTION

The worldwide network of computers commonly referred to as the "Internet" has seen explosive growth in the last several years. The Internet is expected to evolve with the adaptation of new forms of interactive technology applied to the basic Internet infrastructure which consists of many elements, not the least of which are the Web browser and Web page.

Groups of Web pages, forming Web sites, are evolving to a high level of sophistication at a staggering rate. Small to large corporations are taking advantage of this trend, and electronic commerce (E-Commerce), that is, business transactions taking place over the Internet is advancing at a rapid pace. It is highly desirable for those who would like to carry out commerce on the Internet to have a very sophisticated Web site that can perform numerous functions and services to an increasingly sophisticated class of Web site visitors. Such Web sites may desirably include such information services as searchable databases for price, stock, shipping, etc.; product information; competitive comparisons, and so forth.

In order for such information services to be successfully communicated to potential customers, it is imperative to garner the interest of large numbers of Internet users. As with more traditional forms of commerce, advertising plays an important role in attracting customers. Accordingly, what is needed is economical, yet effective, advertising and publicity in order to attract the interest of Internet users.

A recent advance in Web site technology is the addition of streaming media, as well as other more sophisticated functional enhancements, to Web sites. The concept of streaming media is defined broadly as audio and video being delivered to a Web site visitor in packets over the Internet. The streaming media can be delivered so quickly that audio sounds and/or graphic images can be heard and seen almost immediately, comparable in quality to commercial, over-the-air radio or television. Some examples of streaming media include banners, informational feeds using a "marquee", audio based commercials, and so forth.

Unfortunately, it is expensive to add such enhancements to Web sites. Bandwidth costs for delivering streaming media may be prohibitively expensive. In addition, there are problems associated with the complexity of producing the streaming media that is to be "broadcast" over the Web sites, and licensing of the streaming media if it is proprietary.

A typical example of adding function to a Web site is the addition of an "affiliate" program. An affiliate program, provided by a third party may be desired by the Web site developer to add functionality to their Web site for the purpose of enhancing the appeal of the site or for revenue sharing in which they will receive a percentage of sales. In order to obtain such an affiliate program, the Web site developer may be required to register with the supplier of the affiliate program in order to obtain and execute the affiliate program in connection with his/her Web site. Unfortunately,

2

such a registration process typically requires the Web site developer to fill out lengthy on-line electronic forms. Such forms may be cumbersome and so frustrating, that filling out such forms leads to their abandonment on the part of the Web site developer. If the Web site developer successfully manages to register, the Web site developer must then wait for the implementing code for the affiliate program to be e-mailed to him/her. Once the Web site developer receives the implementing code, the code is then copied and pasted onto the HyperText Markup Language (HTML) for the Web site where desired.

Unfortunately, universal capability with the Web browsers that subsequently access the Web site with the enhanced function provided by the affiliate program is limited. That is, even though a Web site developer has successfully added the implementing code for the affiliate program, all Web browsers accessing the Web site may not be able to interpret the affiliate program and the Web site visitor may not be able to experience the added function.

### SUMMARY OF THE INVENTION

Accordingly, it is an advantage of the present invention that a method and system for adding function to a Web page are provided.

It is another advantage of the present invention that a method and system are provided that are compatible with Web browsers which adhere to the standards for HyperText Transfer Protocol (HTTP).

It is another advantage of the present invention that a method and system are provided that add function to a Web page through an easily distributed software code module.

It is yet another advantage of the present invention that a method and system are provided that deliver services by client demand that are specific to predetermined parameters.

The above and other advantages of the present invention are carried out in one form by a method of operating a computer network to add function to a Web page. The method calls for downloading the Web page at a processor platform. When the Web page is downloaded, automatically executing a first code module embedded in the Web page. The first code module issues a first command to retrieve a second code module, via a network connection, from a server system, and the first code module issues a second command to initiate execution of the second code module at the processor platform.

The above and other advantages of the present invention are carried out in another form by a computer readable code module for adding function to a Web page. The code module is configured to be embedded in the Web page which is generated in a HyperText Markup Language (HTML), and is configured for automatic execution when the Web page is downloaded to a client machine supporting a graphical user interface and a Web browser. The computer readable code module includes means for communicating a Web address of the Web page to a server system via a network connection to initiate a download of a second computer readable code module to the client machine. The computer readable code module further includes means for communicating first information characterizing said Web browser to said server and means for communicating second information, characterizing said client machine to said server. In addition, the computer readable code module includes means for initiating execution of said second computer readable code module following the download of the second computer readable code module and means for providing a comment tag informing the Web browser to ignore the initiating means.

US 6,594,691 B1

3

## BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the present invention may be derived by referring to the detailed description and claims when considered in connection with the Figures, wherein like reference numbers refer to similar items throughout the Figures, and:

FIG. 1 shows a block diagram of a computer network in accordance with a preferred embodiment of the present invention;

FIG. 2 shows an exemplary computer readable code module in accordance with the preferred embodiment of the present invention,

FIG. 3 shows a flow chart of a Web page display process.

FIG. 4 shows an electronic display presenting a Web page including a media appliance metaphor;

FIG. 5 shows a flow chart of a service response provision process;

FIG. 6 shows a registration subprocess of the service response provision process;

FIG. 7 shows a Web address database generated by a server system of the computer network;

FIG. 8 shows a visitor registration subprocess of the service response provision process;

FIG. 9 shows a visitor database generated by the server system of the computer network;

FIG. 10 shows a visitor pre-registration process performed prior to the Web page display process of FIG. 3;

FIG. 11 shows the electronic display presenting the media appliance metaphor detached from the Web page; and

FIG. 12 shows the electronic display presenting another Web page including the media appliance metaphor.

## DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

FIG. 1 shows a block diagram of a computer network 20 in accordance with a preferred embodiment of the present invention. Computer network 20 includes a first processor platform 22, a second processor platform 24, and a server system 26. First processor platform 22, second processor platform 24, and server system 26 are connected together via a network 28. In a preferred embodiment, network 28 is the Internet. However, network 28 can also represent a LAN, a WAN, a wireless cellular network, or a combination of a wireline and wireless cellular network. It should be readily apparent to those skilled in the art that computer network 20 also includes many more processors and server systems which are not shown for the sake of clarity.

First processor platform 22 includes a central processing unit (CPU) 30 and a memory 32. Memory 32 includes a Web page 34 in which a first code module 36 is embedded. A Web address 38 in memory 32 is associated with Web page 34. In a preferred embodiment, Web page 34 is generated in HyperText Markup Language (HTML). HTML is the authoring software language used on the Internet's World Wide Web for creating Web pages.

Web address 38 is a Universal Resource Locator (URL), or a string expression used to locate Web page 34 via network 28. It should be readily apparent to those skilled in the art that first processor platform 22 also includes additional components such as input/output lines, a keyboard and/or mouse, and a display terminal which are not shown for the sake of clarity. In addition, memory 32 also contains additional information, such as application programs, operating systems, data, etc., which also are not shown for the sake of clarity.

4

Second processor platform 24 includes a CPU 40, a memory 42, input/output lines 44, an input device 46, such as a keyboard or mouse, a display device 48, such as a display terminal, and speakers 50. Memory 42 includes Web browser software 52 and a temporary memory 54. A first portion of memory 42 is designated for browser information (BROWSER INFO.) 56, and a second portion of memory 42 is designated for platform information (PLATFORM INFO.) 58. In addition, a third portion of memory 42 is designated for a tracking index 60, or cookie, which will be discussed in detail below. Those skilled in the art will understand that memory 42 also contains additional information, such as application programs, operating systems, data, etc., which are not shown in FIG. 1 for the sake of clarity.

Web browser 52 is software which navigates a web of interconnected documents on the World Wide Web via Internet 28. When a Web site, such as Web page 34, is accessed through Web address 38, Web browser 52 moves a copy of Web page 34 into temporary memory 58. Web browser 52 uses HyperText Transfer Protocol (HTTP) for communicating over Internet 28. In a preferred embodiment, Web browser 52 supports the HyperText Markup Language 1.0 and the Javascript 1.0 standards, such as Netscape 2.0 and above, Internet Explorer 3.0, and above, and the like.

Browser information 56 is information specific to Web browser 52. Browser information 56 includes, for example, make and version of Web browser 52, what plug-ins are currently present, and so forth. Platform information 58 is information specific to second processor platform 24. Platform information 58 includes, for example, make and version of platform 24, make and version of the operating system operating on platform 24, and so forth.

Server system 26 includes a processor (CPU) 62, a memory 64, a database structure 66 having a Web address database 68 and a visitor database 70, and a server structure 72 for accommodating streaming media servers 74 and other media servers 76. Ports 78 are in communication with server structure 72 and Internet 28 and are used by the Transmission Control Protocol/Internet Protocol (TCP/IP) transport protocol for providing communication across interconnected networks, between computers with diverse hardware architectures, and with various operating systems.

Memory 64 includes Web address database instructions 80, visitor database instructions 82, a common gateway interface program 84, code assembler instructions 86, and communication instructions 88. Web address database instructions 80 are executed by processor 62 for maintaining and accessing Web address database 68. Likewise, visitor database instructions 82 are executed by processor 62 for maintaining and accessing visitor database 70. CGI interface program 84 executes functions at server system 26 including among other things, checking if Web site 34 is registered. Code assembler instructions 86 are executed by processor 62 to assemble a second code module 90 which is subsequently communicated to second processor platform 24 through the execution of CGI interface program 84 and communication instructions 88. Second code module 90 is communicated from ports 78 over Internet 28 and downloaded to temporary memory 54 at second processor platform 24.

FIG. 2 shows an example format of first code module 36 in accordance with the preferred embodiment of the present invention. First code module 36 is generated in HTML and embedded in the HTML of Web page 34 (FIG. 1) when a Web page developer designs Web page 34. In a preferred embodiment, first code module 36 is generally distributable. That is, first code module 36 may be distributed via Internet



US 6,594,691 B1

5

28, and copied and pasted into a Web page during Web page development. First code module 36 executes enough functionality to act as a "bootstrap loader" in order to load second code module 90 (FIG. 1) into temporary memory 54 (FIG. 1) of second processor platform 24 (FIG. 1) for subsequent execution.

A first command line (LINE NO. 1) 92 contains an exemplary initialization for a first command 93, i.e., a script, that will activate a Web address 94 for contacting server system 26 (FIG. 1) and calls CGI program 84 into execution. In addition, first command line 92 communicates Web address 38 to server system 26 via a network connection 96 (FIG. 1) over Internet 28. CGI program 84 executes multiple functions at server system 26. For example, CGI program 84 checks to see whether or not Web page 34 is registered. In addition CGI program 84 initiates the downloading of second code module 90 to second processor platform 24. A second command line (LINE NO. 2) 98 terminates the script started in first command line 92.

A third command line (LINE NO. 3) 100 starts a new script. Third command line 100 also contains a comment tag 102 used to allow Web browser 52 to ignore a fourth command line (LINE NO. 4) 104. Fourth command line 104 contains a second command 106 that initiates execution of second code module 90 that was downloaded to temporary memory 54 of second processor platform 24. A fifth command line 108 terminates comment tag 102 and terminates the script begun on third command line 100.

FIG. 3 shows a flow chart of a Web page display process 110. Web page display process 110 is performed by second processor platform 24 to add function, such as streaming media or other media services to Web page 34 when downloaded to second processor platform 24.

With reference to FIG. 4, FIG. 4 shows display device 48 (FIG. 1) presenting Web page 34 with added function, namely with the added function of a media appliance metaphor 111 in response to the activities carried out in connection with Web page display process 110.

Media appliance metaphor 111 is a software device that exists in the realm of electronic communication and has a counterpart in the real world. When displayed with Web page 34 on display device 48 of second processor platform 24, media appliance metaphor 111 is a graphic representation of something that looks and behaves like a media appliance. In the exemplary embodiment, media appliance metaphor 111 represents a radio image. Other examples of media appliance metaphors include television images, computer images, computer game toy images, and so forth. When applied to Web page 34, media appliance metaphor 111 gives the visitor to Web page 34 the impression that they already know how to use the device because it looks and acts like something that they are already familiar with.

Metaphors take any form desired for which practical programming constraints can be met. This includes, but is not limited to interactive video games, network games, network information appliances such as web based telephones or call centers, and notification service appliances, like beepers. First code module 36 (FIG. 1) used to apply the metaphor on a Web page is a universal program interface, and acts as a bootstrap loader capable of retrieving and executing programs suitable for such a purpose.

Although the present invention is described in connection with the presentation of media appliance metaphor 111 as applied to Web page 34, it need not be limited to such a media appliance metaphor. Rather, first code module 36 (FIG. 2) can be embedded in a Web page to be executed by

6

a visiting processor platform in order to execute other code modules not associated with media appliance metaphors.

With reference back to FIG. 3, Web page display process 110 begins with a task 112. Task 112 causes Web browser 52 to download Web page 34 at second processor platform 24. In other words, Web browser 52 moves a copy of Web page 34, with the embedded first code module 36 into temporary memory 54 (FIG. 1) of second processor platform 24.

When Web page 34 is downloaded at second processor platform 24 in task 112, a task 114 is performed. Task 114 causes Web browser 52 to automatically execute first code module 36 embedded in Web page 34, a copy of which is now stored in temporary memory 54.

Following task 114, a task 116 is performed. At task 116, first code module 36 executes first command line 92 (FIG. 2) to retrieve second code module 90 by issuing first command 93 to activate Web address 94, contact server system 26 (FIG. 1), and call CGI program 84 into execution.

A task 118 is performed in connection with task 116. Task 118 causes second processor platform 24 to communicate Web address 38 to server system 26 through the execution of first command line 92, as discussed previously.

Next, a task 120 is performed. Like task 118, task 120 causes second processor platform 24 to communicate browser information 56 (FIG. 1) and platform information 58 (FIG. 1), through the execution of first command line 92, to server system 26. Following task 120, second processor platform 24 performs additional activities (not shown) pertinent to the downloading and presentation of Web page 34 on display device 48 (FIG. 1). Furthermore, as indicated by ellipses following task 120, and relevant to display process 110, second processor platform 24 awaits communication from server system 26 before display process 110 can proceed.

FIG. 5 shows a flow chart of a service response provision process 122 performed by server system 26 (FIG. 1) in response to display process 110 (FIG. 3). Process 122 begins with a task 124. Task 124 causes processor 62 (FIG. 1) of server system 26 to receive first command 93 (FIG. 3).

In response to receipt of first command 93 in task 124, a task 126 is performed. At task 126, server system 26 receives Web address 38 communicated by second processor platform 24 at task 118 (FIG. 3) of display process 110 (FIG. 3).

Following task 126, a query task 128 is performed. At query task 128, server system 26 determines if Web page 34 located by Web address 38 is previously registered. That is, processor 62 executes a portion of Web address database instructions 80 to access Web address database 68 in order to locate an entry in Web address database 68 corresponding to Web address 38.

When processor 62 determines that there is no entry in Web Address database 68 for Web address 38, process 122 proceeds to a task 130. Task 130 causes processor 62 of server system 26 to perform a registration subprocess.

FIG. 6 shows a registration subprocess 132 performed in response to task 130 of service response provision process 122 (FIG. 4). Registration subprocess 132 is performed by server system 26 to register Web page 34 with the controlling entity of server system 26. In addition, registration subprocess 132 is performed to determine a service response (discussed below) for Web page 34.

Registration subprocess 132 is performed automatically the first time that Web page 34 is downloaded at a processor platform. Desirably, registration subprocess 132 is invoked immediately following the design of Web page 34 by a Web

US 6,594,691 B1

7

page developer. For example, following the design of Web page 34, the Web page developer may download Web page 34 at a processor platform to review the graphical, textual, and audio content of Web page 34 before Web page 34 becomes generally accessible by visitors.

When query task 128 determines that there is no entry in Web address database 68 for Web address 38 (FIG. 1), server system 26 may schedule a time to perform registration subprocess 132. Alternatively, registration subprocess 132 may be performed at task 130 (FIG. 4) immediately upon acknowledgment that there is no entry in Web address database 68 (FIG. 1).

Registration subprocess 132 begins with a task 134. Task 134 causes server system 26 (FIG. 1) to retrieve Web page 34. Task 134 may also cause server system 26 to retrieve Web pages (not shown) that are nested in association with Web page 34.

In response to task 134, a task 136 is performed. Task 136 causes processor 62 of server system 26 execute a portion of Web address database instructions 80 to extract information content of Web page 34. The information content of Web page 34 is derived from all characters and words that are written on Web page 34, and that are publicly accessible. The information content may then be reduced by extracting informational metatags, or HTML tags, embedded in Web page 34 that are used to specify information about Web page 34. In particular, the "keyword" and "description" metatags usually contain words and description information that accurately describe Web page 34. Other informational content which may be extracted are links, other URLs, domain names, domain name extensions (such as com, .edu., .jp, .uk, etc.), and so forth.

Following task 136, a task 138 is performed. Task 138 causes processor 62 to archive the information content described in connection with task 136.

In response to extraction task 136 and archival task 138, a task 140 is performed. Task 140 causes processor 62 (FIG. 1) executing Web address database instructions 80 to produce a particular "signature" or profile of Web page 34. This profile is important for determining the nature of the interest by a visitor using second processor platform 24 to display Web page 34 from whence the profile is produced in order to perform a service response (discussed below) related to the profile.

Following task 140, a query task 142 is performed. Query task 142 determines whether or not Web page 34 can be registered. Processor 62 (FIG. 1) may determine that Web page 34 cannot be registered if the information content of Web page 34 is objectionable or otherwise unacceptable to be displayed with added function, i.e., media appliance metaphor 111 (FIG. 4). When query task 142 determines that Web page 34 is not to be registered, subprocess 132 proceeds to a task 144.

Task 144 causes processor 62 (FIG. 2) to form a service response indicating a denial of service. In a preferred embodiment, a desired service response is media appliance metaphor 111 functioning to provide streaming media, in this case music, along with Web page 34. However, with respect to task 144, the service response indicating denial of service may be the media appliance metaphor 111 having a slash through it. Alternatively, the service response may simply be an absence of any media appliance metaphor. Following task 144, subprocess 132 proceeds to a task 146.

Referring to FIG. 7 in connection with task 146, FIG. 7 shows Web address database 68 of server system 26 (FIG. 1). Web address database 68 includes as a minimum, a Web

8

address field 150, a Web page profile field 152, a service response field 154, and a parameter set field 156. Task 146 (FIG. 6) causes processor 62 (FIG. 1) to generate an entry, for example, a first exemplary entry 158, in Web address database 68. Web address field 150 is designated for a Web address, or URL. Profile field 152 contains the profile of the Web address produced in task 140 (FIG. 6) of registration subprocess 132. Service response field 154 is designated for a service response, and parameter set field 156 is designated for parameters used to assemble second code module 90 having the desired service response.

First entry 158 generated in response to task 144 (FIG. 6) includes Web address 38 identified simply as URL 1 in Web address field 150, a profile 160 in profile field 152 associated with URL 1 indicates Web page 34 as being directed toward RECREATION/GOLF. A service response 162 related to profile 160 indicating a denial of service is stored in service response field 154 for entry 158, and a denial content parameter set 164 associated with service response 162 are used to form an audible, visual, or other presentation of denial service response 162.

Referring back to query task 142 (FIG. 6) of registration subprocess 132, when query task 142 determines that Web page 34 is registered, subprocess 132 proceeds to a query task 166. At query task 166, processor 62 (FIG. 1) may execute a portion of Web address database instructions 80 to determine if a service response for Web page 34 is to be customized. That is, the Web page developer of Web page 34 has the option of customizing media appliance metaphor 111 (FIG. 4). Such customization may include, but is not limited to music formats tailored to fit the profile, or personality, of Web page 34, the appearance of metaphor 111, the names and formats of the radio channels, the banners that are displayed, the specific type of informational feeds, and so forth.

When processor 62 determines that the service response is to be customized, subprocess 132 proceeds to a task 168. At task 168, processor 62 (FIG. 1) establishes a parameter set for customization of media appliance metaphor 111 to be applied to Web page 34. The custom metaphor is defined by the parameter set. Establishment of the parameter set may be performed through a query exercise performed between server system 26 and the Web page developer of Web page 34. Customization can include references to commercials targeted to Web page 34, custom configuration data, custom Web page metaphor preferences, Web page owner preferences, and so forth.

In response to task 168, a task 170 is performed. Task 170 causes processor 62 to form a service response indicating conditional service, i.e., presentation of media appliance metaphor 111 that has been customized as a result of the activities associated with task 168. Following task 170, registration subprocess 132 proceeds to task 146 for generation of an entry in Web address database 68 (FIG. 7) to store the service response in association with the Web address.

Referring momentarily to FIG. 7, Web address database 68 includes a second exemplary entry 172. Second entry 172 generated in response to task 170 (FIG. 6) includes a Web address 38 in Web address field 150 identified simply as URL 2. A profile 174 in profile field 152 associated with URL 2 indicates Web page 34 as being directed toward TEXAS COOKING. A service response 176 related to profile 174 indicating conditional service is stored in service response field 154 for entry 172, and a conditional content parameter set 178 associated with conditional service



US 6,594,691 B1

9

response 176 is used to form an audible, visual, or other presentation of conditional service response 176.

With reference back to registration subprocess 132 (FIG. 6), when processor 62 determines at query task 166 the service response is not to be customized, registration subprocess 132 proceeds to a task 180. Task 180 causes processor 62 to form a service response indicating a predetermined, or default, service. Such a service response is determined by the entity controlling server system 26 (FIG. 1). In task 180, the controlling entity can determine the look and feel of media appliance metaphor 111 (FIG. 4), the particular audio format to be used with media appliance metaphor 111, for example a particular music type, the controls available to a visitor to Web page 34, and so forth.

Following task 180, subprocess 132 proceeds to task 146 where an entry is generated in Web address database 68 (FIG. 7) to store the service response in association with the web address. Again referring to Web address database 68 (FIG. 7), Web address database 68 includes a third exemplary entry 182. Third entry 182, generated in response to task 180 (FIG. 6), includes Web address 38 in Web address field 150 identified simply as URL 3. A profile 184 in profile field 152 associated with URL 3 indicates Web page 34 as being directed toward WEDDING. A service response 186 indicating a predetermined service is stored in service response field 154 for entry 182, and a predetermined content parameter set 188 associated with service response 186 is used to form an audible, visual, or other presentation of predetermined service response 186.

Following task 146 and the formation of service response 162 indicating denial of service, the formation of service response 176 indicating conditional service, or the formation of service response 186 indicating predetermined service, Web page 34 is registered, and subprocess 132 exits.

Referring back to service response provision process 122 (FIG. 5) following task 130 in which registration subprocess 132 (FIG. 6) has been performed, or when query task 128 determines that Web page 34 (FIG. 1) identified by Web address 38 (FIG. 1) has been previously registered, provision process 122 continues with a task 190.

Task 190 causes processor 62 (FIG. 1) to receive browser information 56 (FIG. 1) and platform information 58 (FIG. 1) from second processor platform 24 (FIG. 1). As discussed previously, browser information 56 includes, for example, make and version of Web browser 52, what plug-ins are currently present, and so forth. Platform information 58 includes, for example, make and version of platform 24, make and version of the operating system operating on platform 24, and so forth.

In response to task 190, a query task 192 is performed. Query task 192 causes processor 62 to execute a portion of visitor database instructions 82 (FIG. 1) to determine if there is an entry in visitor database 70 related to browser information 56 and platform information 58. When query task 192 determines that there is no entry in visitor database 70, indicating that a user of second processor platform 24 has not previously downloaded a Web page containing first code module 36, provision process 122 proceeds to a task 194.

Task 194 causes processor 62 to further execute visitor database instructions 82 to perform a visitor registration subprocess. FIG. 8 shows a visitor registration subprocess 196 of service response provision process 122. Visitor registration subprocess 196 is performed for tracking visitors to Web page 34. Visitor registration subprocess 196 generates visitor database 70 containing visitor demographics and interests that may be useful for targeting advertising and tailoring added function to Web pages.

10

Visitor registration subprocess 196 begins with a task 198. Task 198 causes server system 26 (FIG. 1) to apply tracking index 60 to second processor platform 24 via network connection 96. Tracking index 60, also known as a cookie, is a feature of HTTP that allows the entity controlling server system 26 to place information in memory 42 (FIG. 1) of second processor platform 24. Tracking index 60 allows server system 26 to both store and retrieve information on second processor platform 24. Tracking index 60 is persistent, meaning it remains in memory 42 (FIG. 1) of second processor platform 24 for subsequent use by server system 26. Since tracking index 60 is persistent, tracking index 60 can be used by server system 26 to track a visitor, using second processor platform 24, to any Web page that has embedded therein first code module 36.

In connection with task 198, a task 200 is performed. Task 200 causes processor 62 (FIG. 1) to generate an entry in visitor database 70 to store browser information 56 and platform information 58 in association with tracking index 60. Following task 200, visitor registration subprocess exits.

FIG. 9 shows visitor database 70 generated by server system 26 of computer network 20. Visitor database 70 includes as a minimum, a tracking index field 202, a browser ID field 204, a platform ID field 206, and a visitor preferences field 208. Task 200 (FIG. 8) causes processor 62 (FIG. 1) to generate a visitor database entry 210, in visitor database 70. Tracking index field 202 is designated for a tracking index, or cookie, such as tracking index 60 identifying second processor platform 24. Browser ID field 204 contains browser information 56 received in task 190 (FIG. 5) of provision process 122. Likewise, platform ID field 206 is designated for platform information 58 received in task 190. Visitor preferences field 208 is designated for an optional visitor specified parameter set 212 assembled in response to a visitor pre-registration process (discussed below).

Referring back to service response provision process 122 (FIG. 5), following task 194 in which visitor registration subprocess 196 is performed or when query task 192 determines that entry 210 (FIG. 9) is present in visitor database 70, process 122 proceeds to a query task 214.

Query task 214 determines if entry 210 includes visitor specified parameter set 212. As mentioned previously, visitor specified parameter set 212 may be present if second processor platform has previously performed a visitor pre-registration process.

FIG. 10 shows a visitor pre-registration process 216 performed prior to invoking Web page display process 110 (FIG. 3). Visitor pre-registration process 216 may be performed by a user of second processor platform 24 (FIG. 1) via an access account (not shown). Visitor pre-registration process 216 allows users to have some preference control over any added function, such as media appliance metaphor 111 (FIG. 4) that they may encounter when downloading Web pages having first code module 36 embedded therein.

Visitor pre-registration process 216 begins with a task 218. Task 218 causes processor 62 (FIG. 1) of server system 26 to receive a request (not shown) to pre-register from second processor platform 24. Such a request may be received over a communication link, such as network connection 96, via Internet 28, following the assignment of an access account to second processor platform 24.

In connection with task 218, a task 220 is performed. Task 220 causes processor 62 to receive browser information 56 and platform information 58 from second processor platform 24 via network connection 96.

Following task 220, a task 222 is performed. In a manner similar to task 198 of visitor registration process 196 (FIG.

US 6,594,691 B1

11

8), server system 26 applies a tracking index or cookie, such as tracking index 60, to second processor platform 24.

Next a task 224 is performed. In task 224, processor 62 and second processor platform 24 perform an interactive process to obtain visitor specified parameters for establishing visitor specified parameter set 212 (FIG. 9). Such visitor specified parameters may include, for example, the appearance of specified metaphors, specific audio channels, format preferences, such as location on the Web page, size, color, and so forth.

Following task 224, a task 226 is performed. Task 226 causes processor 62, through the execution of visitor database instructions 82 (FIG. 1), to generate an entry, such as entry 210 (FIG. 9) in visitor database 70 to store browser information 56 and platform information 58 in association with tracking index 60.

In addition a task 228 is performed in connection with task 226. Task 228 causes processor 62, executing visitor database instructions 82, to append entry 210 with visitor specified parameter set 212, as illustrated in visitor database 70 (FIG. 9). Following task 228, visitor pre-registration process 216 exits.

Referring back to query task 214 of service response provision process 122 (FIG. 5), when processor 62 determines that entry 210 (FIG. 9) includes visitor specified parameter set 212 obtained through the execution of visitor pre-registration process 216 (FIG. 10), process 122 proceeds to a task 230.

Task 230 causes processor 62 to access Web address database 68 to amend a service response in service response field 154 (FIG. 7) to indicate a visitor specified conditional service is to be provided for second processor platform 24. Referring momentarily to Web address database 68 (FIG. 7), database 68 includes a fourth exemplary entry 232 for a Web address 38 identified simply as URL 4 in Web address field 150, a profile 234 in profile field 152 associated with URL 4 indicates Web page 34 as being directed toward FOOTBALL. Service response 186 indicating predetermined service is entered in service response field 154 for fourth entry 232, and predetermined content set 188 associated with service response 186 is entered in parameter set field 156.

In response to task 230, service response field 154 also includes a flag 236 associated with tracking index 60 indicating that predetermined service response 186 is amended to conditional service response 176 for second platform 24. Flag 236 indicates to processor 62 to access visitor preferences field 208 (FIG. 9) of visitor database 70 for visitor specified parameter set 212. Although, fourth exemplary entry 232 is shown having a predetermined service response 186, it should be readily understood that the service response may be a conditional response 176 (FIG. 7) in which the Web page designer has customized metaphor 111 (FIG. 4) during registration subprocess 132 (FIG. 6).

With reference back to process 122 (FIG. 5) following task 230 or when query task 214 determines that entry 210 (FIG. 9) of visitor database 70 does not include visitor specified; parameter set 212, process 122 proceeds to a task 238.

Task 238 causes processor 62 to execute code assembler instructions 86 (FIG. 1) to assemble second code module 90. Second code module 90 is assembled by accessing the predetermined one of denial of service response 162 (FIG. 7), conditional service response 176 (FIG. 7), and predetermined service response 186 (FIG. 7) from Web address database 68. In addition, second code module 90 is assembled in response to browser information 56 and plat-

12

form information 58. In other words, second code module 90 is assembled to include the service response and to work with any combination of browser/platform systems.

This feature eliminates the need for an affiliate program to be hard coded, installed onto Web page 34, then tested and debugged by programmers. In addition, since second code module 90 is assembled in response to browser information 56, second code module 90 is compatible with Web browser 52 (FIG. 1) used by second processor platform 24 (FIG. 1).

Second code module 90 may also include another Web address 240, represented in parameter set field 156 of second entry 175 of Web address database 68 (FIG. 7). In this exemplary scenario, the media source (audio, video, graphics, banners, informational feed, etc.) originates from a platform (not shown) connected through Internet 28 (FIG. 1) whose location is specified by Web address 240.

Following assembly of second code module 90 in task 238, a task 242 is performed by server system 26. Task 242 causes processor 62 through the execution of CGI program 84 (FIG. 1), to communicate second code module 90 to second processor platform 24 via network connection 96. In addition, through the execution of communication instructions 88 (FIG. 1) and the execution of appropriate command and control protocols, processor 62 manages servers 72 (FIG. 1) in order to direct information content from the media source having Web address 240 to second processor platform 24.

Referring to Web page display process 110 (FIG. 3), display process 110 performs a task 244. Task 244 is complementary to task 242 of provision process 122. That is, as server system 26 communicates second code module 90 to second processor platform 24, task 244 causes platform 24 to receive, via network connection 96 (FIG. 1), second code module 90. Second code module is subsequently stored in temporary memory 54 (FIG. 1) of second processor platform 24.

Following receipt of second code module 90, process 110 proceeds to a task 246. Task 246 causes Web browser 52 (FIG. 1) to execute third command line 100 (FIG. 2) of first code module 36 containing comment tag 102. In addition, task 246 causes Web browser 52 to execute fourth command line 104 (FIG. 2) of first code module 36 issuing second command 106 to initiate the execution of second code module 90.

In response to issuing second command 106 in task 246, a task 248 is performed. Task 248 causes Web browser 52 to execute second code module 90.

In response to task 248, a task 250 is performed. Task 250 causes media appliance metaphor 111 (FIG. 4) to be applied to Web page 34 for display at display device 48 (FIG. 1). Of course, as discussed previously, if the service response is denial of service response 162, media appliance metaphor 111 may be presented with a slash through it or may be absent from Web page 34.

Referring to FIG. 4, the service response is media appliance metaphor 111 presenting a radio image. Through media appliance metaphor 111, streaming audio in the form of a radio channel 252 playing country music is provided and presented through speakers 50 (FIG. 1). Country radio channel 252 enhances the appeal of Web page 34 through an audio experience that compliments Web page 34 whose information content involves Texas Cooking. In connection with music provided through radio channel 252, commercials may be aired that are related to the information content of Web page 34. Such commercials may include content relevant to Texas cooking, for example, food items, antacids,

US 6,594,691 B1

13

barbecues, and so forth. Thus, metaphor 111 is able to deliver targeted advertising to a visitor accessing Web page 34.

Metaphor 111 also includes additional controls. For example, a drop down menu 254 is provided for selection of a different radio channel. In addition, a control button 256 allows a user to forward and reverse radio channel 252, another control button 258 allows a user to play or pause radio channel 252, and a volume slide 260 allows a user to adjust the volume of radio channel 252. An arrow image 262 included in metaphor 111 activates a portable mode (discussed below).

In response to the display of metaphor 111 in task 250, a query task 264 is performed. Query task 264 causes second processor platform 24, operating through Web browser 52, to determine if a command is detected to detach metaphor 111 from Web page 34 in order to activate a portable mode. A portable mode may be selected when a user clicks on arrow image 262. When task 252 determines that the portable mode has been selected process 110 proceeds to a task 266.

Task 266 causes second processor platform 24 to display metaphor 111, in a portable mode, on a refreshed display. FIG. 11 shows electronic display 48 presenting media appliance metaphor 111 detached from the Web page 34 and appearing in a portable mode 268. In an exemplary embodiment, when arrow image 262 is clicked, metaphor 111 changes in appearance to portable mode 268. This change of appearance may reflect a predetermined response by server system 26 or visitor specified preferences set in visitor pre-registration process 216 (FIG. 10).

FIG. 12 shows electronic display 48 presenting a new Web page 270 downloaded at second processor platform 24 and including media appliance metaphor 111 in portable mode 268. Thus, although Web page 34 (FIG. 11) is no longer being display on electronic display 48, a user of second processor platform is still able to enjoy the information content supplied by metaphor 111.

Following task 266 and when query task 264 determines that metaphor 111 is not to be detached from Web page 34, a query task 272 is performed. Query task 272 determines if display of metaphor 111 is to be terminated. Metaphor 111 may be terminated when a user of second processor platform 24 does not detach metaphor 111 from Web page 34 and downloads a subsequent Web page. In another exemplary scenario, second processor platform 24 may be voluntarily or involuntarily disconnected from server system 26 through the execution of fifth command line 108 (FIG. 2) of first code module 36 terminating second command 106 (FIG. 2). In yet another exemplary scenario, metaphor 111 may be terminated when in portable mode 268 by clicking on the close window control, such as an X symbol 274 (FIG. 12).

When query task 272 determines that metaphor 111 is not to be terminated, program control loops back to task 250 to continue display of metaphor 111. However, when query task 272 determines that metaphor 111 is to be terminated process 110 proceeds to a task 276.

Task 276 causes second processor platform 24 to discontinue the display of metaphor 111 on display device 48. Following task 276, process 110 exits.

Referring to service response provision process 122 (FIG. 5), processor 62 (FIG. 1) of server system 26 performs query task 278. Query task 278 is complementary to query task 272 of display process 110. That is, processor 62 monitors for the termination of metaphor 111 in query task 272 and determines at query task 278 whether service should continue.

14

Communication instructions 88 (FIG. 1) executed by processor 62 includes a timing parameter, or clock, (not shown) that is started to allow for a continuous periodic check for continuation of service. In query task 278, when service is to continue, process 122 proceeds to a task 280. Task 280 causes server system 26, through the continued execution of communication instructions 88 at processor 62, to continue directing streaming media associated with metaphor 111 to second processor platform 24. Following task 280, process 122 loops back to query task 278 to continue the periodic check for continuation of service.

When query task 278 determines that service is to be discontinued, process 122 proceeds to a task 282. Task 282 causes server system 26 to terminate services. That is, task 282 causes server system 26 to discontinue directing streaming media associated with metaphor 111 to second processor platform 24. Following task 282, process 122 exits.

In summary, the present invention teaches of a method and system for adding function, such as streaming media or other media services to a Web page, through the implementation of a simple code module embedded in the HTML of the Web page. The code module is compatible with Web browsers which adhere to the standards for HyperText Transfer Protocol (HTTP) because it is implemented using a common subset of the current HTML standard command set. In addition, the code module is easily distributed through the Internet, and is readily copied and pasted into a Web page during Web page development activities, and undergoes automatic execution and registration with minimal effort by the Web page developer. The present invention is able to tailor the added function based on information about the Web page in which it is embedded and based on visitor specified preferences.

Although the preferred embodiments of the invention have been illustrated and described in detail, it will be readily apparent to those skilled in the art that various modifications may be made therein without departing from the spirit of the invention or from the scope of the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than restrictive sense. Furthermore, although the present invention is described in connection with a media appliance metaphor for providing streaming audio, this is not intended to be limiting. For example, the metaphor may providing streaming video and other multimedia communication formats.

What is claimed is:

1. A method of operating a computer network to add function to a Web page comprising the steps of:

downloading said Web page at a processor platform, said downloading step being performed by a Web browser; when said Web page is downloaded, automatically executing a first code module embedded in said Web page;

said first code module issuing a first command to retrieve a second code module, via a network connection, from a server system;

receiving, at said server system, first information characterizing said Web browser in response to said executing step;

receiving, at said server system, second information characterizing said processor platform in response to said executing step;

storing said first and said second information in a visitor database of said server system, said first and said second information being associated with a tracking index; assembling, at said server system, said second



US 6,594,691 B1

15

code module, said second code module containing a service response related to said Web page; said second code module being responsive to said first and second information;

downloading, in response to said first command, said code module to said processor platform; and

said first code module issuing a second command to initiate execution of said second code module at said processor platform.

2. A method as claimed in claim 1 wherein said Web browser employs HyperText Transfer Protocol (HTTP) and said first code module is generated in a HyperText Markup Language (HTML).

3. A method as claimed in claim 2 wherein said Web page is generated in said HTML, and said first code module includes a comment tag informing said Web browser to ignore said second command.

4. A method as claimed in claim 1 wherein said method further comprises the steps performed by said server system of:

receiving a Web address of said Web page;

determining if said Web page is registered with said server system; and

when said Web page is not registered, performing a registration of said Web page.

5. A method as claimed in claim 4 wherein said performing step comprises the steps of:

receiving said Web page at said server system;

extracting informational content of said Web page;

archiving said informational content of said Web page; and

producing a profile of said Web page in response to said extracting and archiving steps.

6. A method as claimed in claim 5, wherein said service response is related to said profile of said Web page, further comprising the steps of:

storing said service response in association with said Web address;

accessing said service response when said first code module issues said first command.

7. A method as claimed in claim 1 wherein said second code module includes a service response indicating a denial of service.

8. A method as claimed in claim 1 wherein said second code module includes a service response indicating a conditional service.

9. A method as claimed in claim 1 wherein said second code module includes a service response indicating a pre-determined service.

10. A method as claimed in claim 1 further comprising the steps of:

applying said tracking index to said processor platform in response to said first and second information; and

using said tracking index at said server system to track and identify said processor platform.

11. A method as claimed in claim 10 further comprising the steps of:

appending visitor specified parameters to a visitor database entry for said first and said second information associated with said tracking index;

executing said second code module in response to said second command; and

presenting, at said processor platform, a service response having a conditional service characterized by said visitor specified parameters.

16

12. A method as claimed in claim 11 wherein prior to said downloading step, said method further comprises the steps of:

registering said first and second information characterizing said Web browser and said processor platform in said visitor database; and

establishing said visitor specified parameters.

13. A method as claimed in claim 1 further comprising the steps of:

executing said second code module in response to said second command; and

presenting a service response upon execution of said second code module.

14. A method as claimed in claim 13 wherein said service response is a metaphor, and said method further comprises the step of displaying said metaphor in connection with said Web page on said processor platform.

15. A method as claimed in claim 14 further comprising the step of customizing said metaphor to include a parameter set relevant to said Web page, said customized metaphor describing a conditional service presented upon execution of said second code module.

16. A method as claimed in claim 14 further comprising the steps of:

detaching said metaphor from said Web page; and

displaying said metaphor disassociated from said Web page.

17. A method as claimed in claim 13 further comprising the step of terminating said presenting step upon detection, at said server system, of a terminate service response indicator from said processor platform.

18. A method as claimed in claim 1 further comprising the steps of:

executing said second code module in response to said second command, said second code module including a Web address for a second Web page;

downloading information content from said second Web page at said processor platform; and

presenting said information content in a service response at said processor platform.

19. A computer readable code module for adding function to a Web page, said code module configured to be embedded in said Web page generated in a HyperText Markup Language (HTML) and configured for automatic execution when said Web page is downloaded to a client machine supporting a graphical user interface and a Web browser, said computer readable code module including:

means for communicating a Web address of said Web page to a server system via a network connection to initiate a download of a second computer readable code module to said client machine;

means for communicating first information characterizing said Web browser to said server system;

means for Communicating second information characterizing said client machine to said server system; means for assembling, at said server system, said second computer readable code module, said second computer readable code module containing a service response related to said Web page, said second computer readable code module being responsive to said first and second information;

means for downloading said second computer readable code module to said client machine;

means for initiating execution of said second computer readable code module following said download of said second computer readable code module; and

US 6,594,691 B1

17

means for providing a comment tag informing said Web browser to ignore said initiating means.

20. A computer readable code module as claimed in claim 19 wherein said code module is generated in said HTML.

21. A computer network comprising a first processor platform for maintaining a Web page accessible through a Web address, said Web page including a first code module embedded therein, and a second processor platform in communication with said first processor platform via a network connection, said second processor platform supporting a Web browser, said Web browser being configured to download said Web page and execute said first code module, wherein:

said first code module issues a first command to retrieve a second code module; and

said computer network further comprises a server system in communication with said second processor platform for receiving said first command, said server system including:

a database having stored therein a service response in association with said Web address;

a processor, in communication with said database, for assembling said second code module having said service response; and

means for communicating said second code module to said second processor platform, such that upon retrieving said second code module, said first code module issues a second command to initiate execution of said second code module at said second processor platform.

22. A computer network as claimed in claim 21 wherein said server system further comprises a memory element accessible by said processor, said memory element having instructions stored therein which, when executed by said processor, cause said processor to access said database to locate an entry in said database for said Web address, and when said entry is absent, said instructions cause said processor to receive said Web page, extract informational content of said Web page, archive said informational content in said database, and produce a profile of said Web page.

23. A computer network as claimed in claim 22 wherein said instructions further cause said processor to generate said entry for said Web address in said database, said entry including said service response related to said profile.

24. A computer network as claimed in claim 23 wherein said service response is a metaphor, and said instructions further cause said processor to establish a parameter set in response to said profile, said parameter set defining said metaphor.

18

25. A computer network as claimed in claim 21 wherein said server system further comprises:

means for receiving, from said second processor platform, first information related to said Web browser and second information related to said second processor platform, said processor being in communication with said receiving means; and

a memory element accessible by said processor, said memory element having instructions stored therein which, when executed by said processor, cause said processor to assemble said second code module in response to said first and second information.

26. A computer network as claimed in claim 21 wherein said server system further comprises:

means for receiving, from said second processor platform, first information related to said Web browser and second information related to said second processor platform, said processor being in communication with said receiving means;

a visitor database; and

a memory element accessible by said processor, said memory element having instructions stored therein which, when executed by said processor, cause said processor to apply a tracking index to said second processor platform for subsequent access by said server system and to generate an entry in said visitor database, said entry including said first and second information related to said tracking index.

27. A computer network as claimed in claim 26 wherein said server system further comprises:

an input element for receiving visitor specified parameters; and

means for amending said entry in said visitor database in response to said visitor specified parameters.

28. A computer network as claimed in claim 21 wherein said service response includes a second Web address for a second Web page, and said second code module includes said second Web address for subsequent download of information content from said second Web page by said Web browser of said second processor platform when said first code module issues said second command.

\* \* \* \* \*

UNITED STATES PATENT AND TRADEMARK OFFICE  
**CERTIFICATE OF CORRECTION**

PATENT NO. : 6,594,691 B1  
DATED : July 15, 2003  
INVENTOR(S) : McCollum et al.

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 7,

Line 32, insert the words -- (such as .com, .edu, .jp, .uk, etc.), -- and delete the words -- (such as com, .edu., .jp, .uk, etc.), -- after the words "domain name extensions" and before the words "and so forth."

Column 11,


Line 58, insert the words -- specified parameter -- and delete the words -- specified; parameter -- after the words "include visitor" and before the words "set 212, process"

Column 13,

Line 45, insert the words -- subsequent web page. -- and delete the words -- subsequent. Web page. -- after the words "and downloads a" at the end of the sentence.

Signed and Sealed this

Fourteenth Day of October, 2003

A handwritten signature in black ink, appearing to read "James E. Rogan", with a horizontal line drawn underneath it.

JAMES E. ROGAN  
*Director of the United States Patent and Trademark Office*